Universiteit Antwerpen

Faculteit Wetenschappen
Informatica

Implementation and Evaluation of timeliness in Wireless Networks: to be or not to be - on time

Implementatie en Evaluatie van tijdigheid in Draadloze Netwerken: (op tijd) te zijn of niet te zijn

Proefschrift voorgelegd tot het behalen van de graad van Doctor in de Wetenschappen aan de Universiteit Antwerpen te verdedigen door

Wim TORFS

Promotor: Prof. Dr. Chris Blondia

Antwerpen, 2019

Acknowledgments

I would like to thank Prof. Dr. Chris Blondia who provided me with the opportunity to obtain a PhD. In particular I would like to thank him for giving me the freedom to operate and the interesting discussions.

I would also like to thank Dr. S.A. Romaszko for sharing her expertise regarding Cognitive Radio and Rendezvous protocols. Her knowledge regarding these topics was paramount in performing the relevant research.

I would also like to thank the members of the jury for providing constructive comments and their positive encouragements.

My thanks towards the formerly PATS, and now MOSAIC, administrators, in particular Johan Bergs, who helped me numerous times with my VPN issues.

Finally, I would like to thank my wife, Sylwia, and my daughter, Izabela, for being patient and understanding of all the time spent working.

Nederlandse samenvatting –Summary in Dutch–

Draadloze communicatie is een gebied met vele toepassingsmogelijkheden dat snel evolueert. Voor elk soort van applicatie is er wel een bepaald netwerk dat ervoor kan dienen. Binnen elk van die netwerken bevinden er zich verscheidene onderdelen waarbinnen onderzoek uitgevoerd wordt. Slechts een klein gedeelte van het beschikbare onderzoek is in dit werk beschreven, dat onderzoek uitgevoerd binnen drie verschillende netwerk types bevat, voornamelijk in de context van Medium Access Control (MAC) protocollen. De aangehaalde netwerken bevatten Wireless Sensor Networks, Wireless Local Area Networks en Cognitive Radio Networks.

Wireless Sensor Networks (WSNs) zijn bekend om hun relatief beperkte capaciteiten zowel wat betreft energie als processing power. Vandaar dat vele werken gefocust zijn op de energie efficiëntie van MAC protocollen voor wat betreft sensor netwerken. Een onderscheid kan gemaakt worden tussen een gecontroleerde medium access en een random medium access. Veel van de werken die de laatste methode gebruiken, stellen een verdeling van de beschikbare tijd voor, zodat een deel van de tijd de nodes actief zijn en een deel van de tijd energie kunnen sparen. Evenals de methode van Low Power Listening wordt gebruikt in dergelijke situaties om energie te sparen. Helaas neemt dit niet weg dat idle listening of overhearing nog steeds kunnen gebeuren. Een gecontroleerde medium access biedt de mogelijkheid om de voorgenoemde fenomenen grotendeels te elimineren dankzij de nauwkeurige controle over de transmissies en ontvangsten. Vandaar dat een groot deel van de reeds beschikbare MAC protocollen voor WSNs gebaseerd zijn op Time Division Multiple Access (TDMA). Binnen deze protocollen bestaat nog steeds een grote verscheidenheid wat betreft de methode van slot allocation. Sommige methodes gebruiken een gedistribueerde aanpak, terwijl andere dan weer een gecentraliseerde aanpak verkiezen, sommigen beschouwen zelfs een geladderde slot allocatie, etc. Wat duidelijk is, is dat er weinig werken zijn die een heterogeen network in beschouwing nemen. Met heterogeen wordt een netwerk bedoeld waarin niet elke node evenveel informatie te versturen heeft. Deze thesis neemt dergelijke netwerken wel onder beschouwing en stelt een protocol voor waarin zowel nodes met een grote hoeveelheid data, als

nodes met een kleine hoeveelheid data kunnen co-existeren binnen eenzelfde netwerk. Vaak wordt de nood aan synchronisatie beschouwd als een nadeel van gecontroleerde medium access. Vandaar dat dit werk zich dan ook verdiept in bestaande synchronisatie methodes. Allereerst wordt een theoretische beschouwing gedaan van de protocollen, waarna deze wordt toegepast op hedendaagse sensor netwerk platformen. Gebaseerd op de resultaten van deze analyse is een meer efficiënt synchronisatie protocol gecontrueerd. Om de performance van dit protocol voor wat betreft efficiëntie en stabiliteit te kunnen aantonen, wordt de implementatie ervan en diens resultaten in een Proof of Concept (PoC) beschreven.

Een tweede netwerk dat beschouwd wordt in dit werk is het Wireless Local Area Network. Vanwege een hele reeks redenen zijn de specificaties opgesteld door het Institute of Electrical and Electronics Engineers (IEEE) niet geschikt voor een aantal specifieke doeleinden, zoals bv. Long Range Networks, een gegarandeerde Quality of Service (QoS) voor onder andere multimedia streams of Voice over IP (VoIP) of batterijgevoede bewakingscameras. Het gebrek van bepaalde functionele elementen van de hardware, die compatibel is met de standaard, zorgt ervoor dat een minder dan optimale performance wordt bereikt in dergelijke toepassingsgebieden. Een aantal van de ondervonden problemen zijn onder andere idle listening, het gebrek van QoS garanties, de beperkte tijd voor wat betreft het ontvangen van een Acknowledgement (ACK) of het uitvoeren van een Clear Channel Assessment (CCA). Verscheidene werken hebben onderzocht hoe commercieel beschikbare hardware die voldoet aan de specificaties van de standaard kan gebruikt worden op een niet standaard manier. De onderzochte materie bij de meeste van dergelijke werken houdt in dat een gecontroleerde medium access is geplaatst bovenop de Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) medium access die door de hardware wordt aangeboden. Verscheidene manieren werden gebruikt om de doelstelling te bereiken. Een aantal werken passen de MadWiFi driver [1] aan, oftewel implementeren een laag bovenop deze driver, dewelke reeds een oudere driver is voor legacy hardware. Andere werken bereiken hun doelstelling door een combinatie van user space drivers en kernel modules die de commandos doorgeven. Nog andere werken gebruiken een ander type van hardware en passen de specifieke driver aan, of ze gebruiken de hardware in monitor mode, waardoor de gebruikelijke standaard functionaliteit van de radio wordt opgehoffen.

Ongeacht van de gebruikte methode, hebben de meeste, of zelfs alle, van deze werken hun doelstelling kunnen bereiken dankzij het open source karakter van de Linux kernel. Bij het ontwerpen of wijzigen van delen van de kernel zodat een gecontroleerde medium access mogelijk gemaakt wordt is het belangrijk om de verschillende delen van de kernel te identificeren die mogelijks een invloed kunnen hebben op de precisie waarmee de transmissie gecontroleerd wordt. Een van deze onderdelen is het process scheduling systeem dat bepaalt welke taak als volgende dient te worden uitgevoerd en bepaalt ook mede de onderlinge prioriteit van de verschillende taken. Verder is het gedeelte dat verantwoordelijk is voor het afhandelen van interrupts van belang, aangezien het het normale verloop van de processes beïnvloedt en vaak vandaar uit een tasklet of softirq gescheduled wordt, dewelke een hogere prioriteit hebben dan reguliere processen. De timer functionaliteit biedt een essentiële service van precieze timer interrupts om zodanig de transmissies en ontvangsten nauwkeurig te kunnen controleren. Uiteraard dient ook met het netwerk subsysteem van de Linux kernel rekening gehouden te worden, wat de hardware drivers implementeert, alsook de MAC functionaliteit, de Logical Link Control (LLC) en de hogere lagen met onder meer Internet Protocol (IP), User Datagram Protocol(UDP) en Transport Control Protocol(TCP). Een zekere mate van begrip omtrent deze onderdelen is vereist indien dergelijke tijds gevoelige applicaties worden ontworpen.

Aangezien de meeste hardware chips enkel maar de CSMA medium access methode ondersteunen, moet de werking van de verschillende onderdelen en functionaliteiten van de standaard, die mogelijks een invloed zouden kunnen uitoefenen op de precisie van de transmissie of ontvangst, moeten worden bestudeerd. De Distributed Coordination Function (DCF) is een van de onderdelen die hierop het meeste invloed zal hebben, maar ook met aggregation en het Block Ack mechanisme moet rekening gehouden worden, daar die toch ook een belangrijke invloedssfeer hebben. Om een dergelijke implementatie tot stand te brengen is allereerst een nauwkeurige timer source nodig. Vandaar dat een performance analyse van beschikbare timer sources is uitgevoerd foor een reeks van verschillende omgevingen en parameters. Van verscheidene Linux kernel real-time extensions en een enkel real-time operating system zijn ook mede gebruik gemaakt binnen deze analyse. Op basis van de verkregen resultaten is een van de timers geselecteerd om de precieze transmissie tijden aan te geven. De implementatie toonde nog verscheidene hardware artefacten die een invloed hadden op de nauwkeurigheid, waar voor ieder verschijnsel een oplossing of een workaround is gevonden.

Een derde netwerk dat in rekening wordt gebracht binnen dit werk is het Cognitive Radio Network (CRN). Daar waar al het voor vermelde werk gerelateerd was tot MAC protocollen, gaat dit gedeelte van het werk over een functionaliteit dit wordt uitgevoerd nog voordat de MAC laag aan het werk moet. Een CRN is een netwerk dat specifieke eigenschappen en eisen vertoont, waar aan gehouden moet worden. Bijvoorbeeld, de Primary User (PU) activiteit zou nooit onderbroken of verstoord mogen worden door een Rendezvous protocol. Verder moet er rekening gehouden worden dat een PU plots actief kan worden op het kanaal dat de Secondary Users (SUs) momenteel aan het gebruiken zijn. Vandaar dat het ontwerpen van een Rendezvous protocol de nodige aandacht vereist. Een populaire aanpak is die van channel hopping, waarbij nodes over de verschillende kanalen hoppen volgens een vastgestelde sequentie en proberen andere SUs te detecteren.

Binnen de werken die channel hopping beschouwen bestaat er nog een grote verscheidenheid over de manier waarop het hopping patroon tot stand komt. De aanpakken gaan van het aanmaken van een random hopping patroon, waarbij dan op ieder kanaal een evengrote kans is om een rendezvous te hebben, tot een voorspelbare rendezvous die gebaseerd is op numerieke analyse. Ongeacht van de variëteit van dergelijke protocollen, de meesten nemen het natuurlijke asynchrone karakter van het systeem niet in rekening, terwijl dit net de performance enorm zou kunnen beïnvloeden.

Dit werk stelt een manier voor waar op een efficiënte manier een asynchrone analyse kan uitgevoerd worden van bijna elk Rendezvous protocol. Het resultaat van die analyse toonde dat de werking op een asynchrone manier een verbetering met zich meebrengt voor wat betreft de performance van het Rendezvous protocol. Vandaar dat een extension is ontworpen die een dergelijke asynchrone werking forceert bij eender welk Rendezvous protocol door een grotere slot size te bieden aan kanalen die een hogere prioriteit hebben. Verder is in dit werk een meer geoptimaliseerde extension voorgesteld die een betere verdeling van de slot sizes over de beschikbare kanalen verzorgt. Wanneer deze extension gebruikt wordt in combinatie met een Rendezvous protocol dat specifiek was ontworpen om gebruikt te worden met deze extension, is het duidelijk dat de performance van deze combinatie ver bovenuit de performance van eender welk regulier Rendezvous protocol uitsteekt. Bemerk wel dat de meeste werken enkel maar een analyse of bewijs leveren voor een Rendezvous aan het begin van het hopping patroon. Dit werk neemt alle mogelijke rendezvous volgens dit hopping patroon in rekening en het resultaat is een statistische voorstelling van alle mogelijke rendezvous.

English summary

The area of wireless communications is a broad and fast evolving area. For each application area a specific network type is available. Those network types are composed of several subareas in which research is being performed. A small section of the available research is presented in this work, which discusses three different network types in the context of mostly Medium Access Control (MAC) protocols. The discussion includes Wireless Sensor Networks, Wireless Local Area Networks and Cognitive Radio Networks.

Wireless Sensor Networks (WSNs) are characterized by their low amount of available resources, both in terms of battery power as processing power. Most works therefore focus on the energy efficiency of MAC protocols for sensor networks. The approach taken by the different works can be divided in two main classes, a scheduled approach and an uncoordinated approach. The latter either makes use of duty cycling, or low power listening in order to reduce the energy consumption of the network. However, idle listening and overhearing is not completely eliminated by these methods. A scheduled approach can, thanks to its precise control of the transmissions and receptions, avoid most of the idle listening and overhearing. Therefore, a considerable number of MAC protocols for WSNs are Timer Division Multiple Access (TDMA) based. The slot allocation methods for those protocols come in a wide variety. Some allocation methods use distributed approaches, others a centralized approach, a staggered slot allocation might be considered, etc. However, few works actually consider a heterogeneous network in which the sensor nodes have a different amount of data to transmit. This work proposes a method which ensures that both high throughput and low throughput nodes can exist simultaneously in the same network. Note that an often considered downside of the scheduled medium access is the need for synchronization. This work elaborates on existing synchronization methods, first from a theoretical viewpoint and afterwards applied to common sensor network platforms. Based on the outcome of the analysis, a more efficient synchronization protocol is proposed, which has also been implemented in a Proof of Concept (PoC) to demonstrate its efficiency and stability.

The second type of network considered in this work is the Wireless Local Area

Network. For a wide range of reasons, the Institute of Electrical and Electronics Engineers (IEEE) specifications do not match the requirements of specific target applications, such as Long Range Networks, a guaranteed Quality of Service (QoS) for multimedia streams or Voice over IP (VoIP) or battery powered surveillance cameras. Due to the lack of certain features the default operation of the standard compliant hardware is suboptimal for such applications. The encountered issues include idle listening, the lack of a QoS guarantee, the limited timing constraints for both Acknowledgement (ACK) and Clear Channel Assessment (CCA), etc. Numerous works have been investigating the usage of Commercial Off-the-Shelf hardware (COTS) in a non-standardized manner. The modification for most of these works involves a scheduled medium access on top of the by hardware implemented Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) medium access technology. The modifications were accomplished by means of different approaches. Several works consider the modification of or implementation on top of the MadWiFi driver [1], which is a driver for older hardware. Other works consider a user space driver in combination with custom kernel modules which pass the commands. Yet other works use a different type of hardware or use it in monitoring mode, thereby disabling the usual default functionality of the radio.

Irrespective of the used methodology, most, if not all, of these works were able to complete their target thanks to the open source character of the Linux kernel. Important to consider when designing or modifying a part of the Linux kernel, in order to ensure a scheduled transmission, are the different subsystems which influence the global behavior. The process scheduling for example controls which task is to be scheduled next and determines the priority of the processes. Moreover, the interrupt handling process influences the normal operation of the process scheduling in that the Interrupt Request (IRQ) handler and the often from there scheduled tasklets and softirqs have a higher priority than a regular process. The timer subsystem provides critical functionality of accurate time interrupts in order to schedule the transmission and receptions at precise time intervals. A last subsystem which greatly influences the network operation is the network subsystem of the Linux kernel, which implements the hardware drivers, the MAC protocol, Logical Link Control (LLC), and upper layers such as Internet Protocol (IP), User Datagram Protocol(UDP) and Transport Control Protocol(TCP). A certain comprehension regarding these matters is imperative when designing such time critical application.

Considering only the CSMA access method of the standard is implemented in most hardware chips, the different functionalities which could influence the timing of the transmission or reception needs to be studied in detail. Amongst those is the Distributed Coordination Function (DCF) the one with the largest impact, notwithstanding that aggregation and the Block Ack mechanism also have a significant influence. Note that the implementation of a scheduled transmission requires a precise timer source. Therefore a performance analysis of the available timer sources is done for different environments and parameters. Several Linux kernel real-time extensions and a single real-time operating system are included in the timer source performance analysis. Based on the obtained results, one of the timers is selected as a source for triggering the exact transmission times. The implementation revealed several hardware functions which impacted the accuracy of the transmission time, for each of which a solution or workaround has been found.

The third considered network regards a Cognitive Radio Network (CRN). The previous work was oriented towards MAC protocols or related functionality. The work performed in the CRN considers Rendezvous protocols, which is an operation that should be performed in an earlier stage than the MAC protocol operation. A CRN is a type of network with its own specific properties that should be adhered to. As such, the Rendezvous protocol should not interfere with the operation of the Primary User (PU). Moreover, it should be taken into account that the PU could appear suddenly on the channel used by the Secondary User (SU). Therefore the design of a Rendezvous protocol requires the necessary attention. A popular approach for Rendezvous protocols is the channel hopping method, where nodes hop according to a certain channel sequence and sense the channel for any presence of other SUs.

Within the set of works that consider a channel hopping approach, a wide variety of methodologies exists in determining the hopping pattern. The approaches range from the generation of a random hopping pattern, thereby ensuring a statistical equal distribution of the rendezvous over the number of available channels, towards a most deterministic rendezvous based on numeric system analysis. However, most works do not consider the natural asynchronous character of the system, while this property could influence the performance in a great deal.

This work considers a method to perform an asynchronous analysis in a most efficient manner for any Rendezvous protocol. The outcome of such analysis showed that the asynchronous operation of the system improves the performance of the Rendezvous protocol. Therefore an extension was designed such that the asynchronism is induced on any regular protocol by assigning a larger slot to higher priority channels compared to lower priority channels. Based on the previously designed extension, an improved version has been designed, in order to improve the distribution between the slot sizes over the channel priorities. The performance of the combination with a Rendezvous protocol specifically designed to cooperate with the extension clearly shows a significant improvement compared to regular Rendezvous protocols. Note that while most works relate the performance of their proposed protocol to the start of the hopping pattern, this work considers an analysis of the entire set of possible rendezvous as a result of the generated hopping pattern.

List of publications

1. Wim Torfs, Peter De Cleyn, Chris Blondia, Daan Pareit, Nele Gheysens, Tom Van Leeuwen, Ingrid Moerman, Walter Van Brussel, Ines Clenjans, *"QoS-enabled internet-on-train network architecture: interworking by MMP-SCTP versus MIP"* - in proceedings of the 7th International Conference on ITS Telecommunications, Sophia Antipolis,France, 2007.

2. Wim Torfs, Peter De Cleyn, Chris Blondia, Daan Pareit, Ingrid Moerman, Piet Demeester, *"SCTP as mobility protocol for enhancing Internet on the train"* - in proceedings of the 8th International Conference on ITS Telecommunications, Hilton Phuket, Thailand, 2008.

3. Wim Torfs, Chris Blondia, *"QoS support for a MAC with a TDMA tree topology on the Magnetic Induction Radio IC"* - in proceedings of the 29th Real-Time Systems Symposium (RTSS 2008), Work in Progress session, Barcelona, Spain, 2008.

4. Wim Torfs, Chris Blondia, *"WBAN implementation on Magnetic Induction Radio IC for medical remote monitoring"* - in proceedings of the 1st International Workshop on Medical Applications Networking, Dresden, Germany, 2009.

5. Wim Torfs, Chris Blondia, *"Binary TDMA Schedule by Means of Egyptian Fractions for Real-time WSNs on TMotes"* - in proceedings of Med-Hoc-Net 2010, Juan-les-Pins, France, 2010.

6. Wim Torfs, Chris Blondia, *"Analysis of TDMA scheduling by means of Egyptian Fractions for real-time WSNs"* - in EURASIP Journal on Wireless Communications and Networking 2011.

7. Nicolas Letor, Wim Torfs, Chris Blondia, *"Multimedia multicast performance analysis for 802.11n network cards"* - in proceedings of Wireless Days 2012 (WD 12), 2012.

8. Sylwia Romaszko, Wim Torfs, Sylwia Romaszko, Petri Mähönen, Chris Blondia, *"An analysis of asynchronism of a neighborhood discovery protocol for cognitive radio networks"* - in proceedings of IEEE PIMRC, London, UK, 2013.

9. Sylwia Romaszko,Wim Torfs, Petri Mähönen, Chris Blondia, *"Benefiting from an Induced Asynchronism in Neighborhood Discovery in opportunistic Cognitive Wireless Networks"* - in proceedings of ACM MobiWac, Barcelona, Spain, 2013.

10. Sylwia Romaszko, Wim Torfs, Petri Mähönen, Chris Blondia, *"AND: Asynchronous Neighborhood Discovery protocols for opportunistic Cognitive Wireless Networks"* - in proceedings of IEEE WCNC, Turkey, 2014.

11. Sylwia Romaszko, Wim Torfs, Petri Mähönen, Chris Blondia, *"How to be an efficient Asynchronous Neighbourhood Discovery protocol in opportunistic Cognitive Wireless Networks"* - in International Journal of Ad-hoc and Ubiquitous Computing, Vol. 20, No. 3, 2015.

12. Wim Torfs, Chris Blondia, *"TDMA on commercial of-the-shelf hardware: fact and fiction revealed"* - in International Journal of Electronics and Communications, Vol. 69, No. 5, 2015.

Contents

List of Tables

List of Figures

List of Acronyms

A Ampere

AC Access Category

A-MPDU Aggregate MAC Protocol Data Unit

A-MSDU Aggregate MAC Service Data Unit

AIFS arbitration interframe space

AP Access Point

API Application Program Interface

ARM Advanced RISC Machine

ASAP As Soon As Possible

ASIC Application Specific Integrated Circuit

AV AmpereVolt

BA Block Ack

BAN Body Area Network

BEB Binary Exponential Backoff

BIBD Balanced Incomplete Block Design

BNEP Bluetooth Network Encapsulation Protocol

bps Bits per Second

Bps Bytes per Second

BSS Basic Service Set

C Capacitor

CCA Clear Channel Assessment

CCC Common Control Channel

CDMA Code Division Multiple Acces

CDS Cyclic Difference Set

CFP Contention Free Period

COTS Commercial Off-the-Shelf

CP Contention Period

CPFSK Continuous Phase Frequency Shift Keying

CR Cognitive Radio

CRC Cyclic Redundancy Check

CRN Cognitive Radio Network

CSMA Carrier Sense Multiple Access

CSMA/CD Carrier Sense Multiple Access Collision Detection

CSMA/CA Carrier Sense Multiple Access Collision Avoidance

CTS Clear To Send

CW Contention Window

DA Destination Address

DCF Distributed Coordination Function

DCO Digitally Controlled Oscillator

DCU DCF Control Unit

DIFS Distributed (Coordination Function) Interframe Space

DL Down-Link

DMA Direct Memory Access

DRU	DMA Receive Unit
DS	Direct Spread
DSP	Digital Signal Processor
DSR	Deferred Service Routine
ED	Energy Detect
EDCA	Enhanced Distributed Channel Access
EDF	Earliest Deadline First
ECG	Electrocardiogram
EEVDF	Earliest Eligible Virtual Deadline First
EIFS	Extented Interframe Space
FBWA	Fixed Broadband Wireless Access
FCS	Frame Check Sequence
FDD	Frequency Division Duplex
FDM	Frequency Division Multiplexing
FEC	Forward Error Correction
FHSS	Frequence-Hopping Spread Spectrum
FIFO	First In First Out
FQ	Fair Queuing
FSM	Finite State Machine
FTP	File Transfer Protocol
gcd	greatest common divisor
GPOS	General Purpose Operating System
GPS	Generalized Processor Sharing
GSM	Global System for Mobile Communications

HAL Hardware Abstraction Layer

HCCA HCF Controlled Channel Access

HCF Hybrid Coordination Function

I2C Inter Integrated Circuit

IBSS Independent Basic Service Set

IC Integrated Circuit

IEEE Institute of Electrical and Electronics Engineers

ioctl I/O control

IP Internet Protocol

IPC Inter Process Communication

IRQ Interrupt Request

IRS Identical Row Square

ISR Interrupt Service Routine

LAN Local Area Network

lcm least common multiple

LLC Logical Link Control

LPL Low Power Listening

LS Latin Square

MAC Medium Access Control

MCS Modulation and Coding Scheme

MCU Microcontroller

MLME MAC Layer Management Entity

MOSFET Metal-Oxide-Semiconductor Field-Effect Transistor

MMPDU MAC Management Protocol Data Unit

MPDU MAC Protocol Data Unit

MSDU MAC Service Data Unit

MTTR Maximum Time To Rendezvous

NAPI New Application Interface

NAV Network Allocation Vector

ND Neighborhood Discovery

NIC Network Interface Card

NMOS n-channel MOSFET

NTP Network Time Protocol

OS Operating System

P2P Point to Point

PAN Personal Area Networking

PC Personal Computer

PCB Printed Circuit Board

PCF Point Coordination Function

PCI Peripheral Component Interconnect

PCIe Peripheral Component Interconnect Express

PGPS Packet-by-Packet Generalized Processor Sharing

PHY Physical layer

PIC Programmable Interrupt Controller

PLL Phase Locked Loop

PM Power Management

PoC	Proof of Concept
ppm	parts per million
PSM	Power Save Mode
PWM	Pulse Width Modulation
PU	Primary User
QCU	Queue Control Unit
QoS	Quality of Service
QS	Quorum System
R	Resistor
RA	Receive Address
RAM	Random Access Memory
RC	Resistor and Capacitor
RCP	Rotation Closure Property
RDV	Rendezvous
REM	Radio Environmental Map
RISC	Reduced Instruction Set Computing
RR	Round-Robin
RTDM	Real-Time Driver Model
RTOS	Real-Time Operating System
RTS	Request To Send
RTT	Round Trip Time
SA	Source Address
SFD	Start of Frame Delimiter
SGI	Short Guard Interval

SIFS	Short Interframe Space
SME	Station Management Entity
SMP	Symmetric Multi-Processing
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SSID	Service Set Identifier
SU	Secondary User
TA	Transmit Address
TCP	Transport Control Protocol
TDD	Time Division Duplex
TDM	Time Division Multiplexing
TDMA	Time Division Multiple Access
TIM	Traffic Indication Map
TTL	Time To Live
TTR	Time To Rendezvous
TXOP	Transmission Opportunity
U-APSD	Unscheduled Automatic Power Save Delivery
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
UE	User Equipment
UL	Up-Link
UMTS	Universal Mobile Telecommunications System
uRLLC	ultra Reliable and Low Latency Communication
USRP	Universal Software Radio Peripheral
UTC	Coordinated Universal Time, Temps Universel Coordoné

UTRA	UMTS Terrestrial Radio Access
V	Volt
VoIP	Voice over IP
W	Watt
WBAN	Wireless Body Area Network
WCDMA	Wideband Code Division Multiple Acces
WDS	Wireless Distribution System
WEP	Wired Equivalent Privacy
WFQ	Weighted Fair Queuing
WLAN	Wireless Local Area Network
WME	Wireless Multimedia Extensions
WMM	Wi-Fi Multimedia
WPA2	Wi-Fi Protected Access 2
WSN	Wireless Sensor Network

CHAPTER 1

Introduction

Timeliness is "The fact or quality of being done or occurring at a favorable or useful time" according to Oxford Dictionaries [2]. In other words, timeliness signifies being on time, which is an ever present concept, probably recognizable for everyone. A student who needs to finish a thesis before the deadline, A mom who needs to drive her kids to school, people which need to get to work on time, a demo that needs to be working before the grand presentation, etc. Everyone is familiar with the phenomenon 'being on time' and realizes that it can be sometimes hard to comply to this time demand. In wireless networks time is equally important, perhaps even more so than in our everyday life. One could consider the timeliness issue of communicating nodes similar to the communication between persons. Although people consider it implicitly when talking, timeliness plays a large role during the process. When one person is talking, the other person should be listening. When the timing is not accurately timed, misconceptions or even arguments could arise. Likewise, when the transmission and receptions of network nodes are not accurately timed, messages are either misinterpreted or missed, which could result in catastrophic results in some cases. Note that the transmission and reception timings are the most obvious time restrictions that influence the performance of wireless networks. This work considers not only those obvious restrictions, but also the more subtle variations in timings that could affect the network performance adversely. Note that some concepts will be used during this introduction, such as Synchronization protocols, Rendezvous protocols, etc., which will be explained in one of the next chapters.

As already mentioned, time is rather important in wireless communication. Synchronization protocols enable wireless nodes to use a comparative time reference, such that for example data measurements can be correlated to each other by means of their timestamps. Moreover, the time synchronization enables nodes to coordinate a spectrum access time slot allocation. However, such allocation should take into account amongst others the switching time between radio states, such as idle, transmitting or receiving, and the time required to transmit the preamble when considering implementations on actual hardware. Such physical limitations influence the effective throughput, that is, the number of bytes sent per time unit, and latency, which is defined as the time required for node B to receive a packet originating from node A. Besides the physical limitations, the implementation of a MAC protocol can influence both throughput and latency in a great deal. For example, an efficient TDMA protocol should be able to schedule its time slots and packets in a very precise manner. In order to achieve such precision, a very accurate timer is required. Any deviations in spectrum access time could result in collisions, which is to be avoided in TDMA spectrum access mode. Therefore, the slot timings should take into account the expected deviations, such that no collisions occur. As such, large deviations lower the effectively achievable throughput and increase the minimal latency. Therefore, all functionality which could possibly have an impact on the transmission time is to be disabled or its effect is to be minimized. Besides the throughput and latency, there exists also other metrics which are related to time. In Rendezvous protocols, the Time-To-Rendezvous is one of the most prominent metrics according to which its performance is evaluated. The more time it takes to have a Rendezvous, the fewer Rendezvous opportunities arise during a specific time interval.

This work, which includes mostly research focused on the MAC layer of wireless communication networks, considers all previously mentioned cases where time proves to have a significant impact on the efficiency. More specifically, this work focuses on low latency communication where the overall efficiency is optimized. Note that the concept of time is not dedicated to one single research area. Therefore, this work covers various research areas concerning wireless communications, such as Wireless Sensor Networks (WSNs), Wireless Local Area Networks (WLANs) and Cognitive Radio Networks (CRNs), implying the importance of time in all those disciplines. Each of the disciplines is shortly discussed in the following paragraphs, after which the specific properties that a MAC layer should adhere to are mentioned. Note that this introduction employs several concepts which are left unexplained in this chapter, since they are discussed in detail in the respective chapters.

A WSN can be considered as a network of a large number of small devices, each of them performing a small portion of the work. The devices, also called sensor nodes, usually are small and constrained in terms of resources such as battery power, processing power, memory, etc. On the other hand, the cost of such devices is usually kept low, making it feasible to include a lot of such devices in the network, from a budgetary point of view. A sensor node can be classified as an actuator, that is, a node which enforces some action on its environment, or as a sensor, i.e., a device which gathers information about its environment. In this work, only sensor nodes that gather information are being considered. The gathered information is usually forwarded to the sink node and thanks to the multitude of sensor nodes collecting and forwarding information, a very precise view of the measured parameters can be constructed. Therefore, despite the limited available resources of the devices, they provide a significant added value with respect to information gathering while organized in a network. Besides the obvious appeal of WSNs, most of the sensor node hardware also allows low level programming and configuration. As such, the research is not limited to strictly standardized hardware and new designs can be verified in a straightforward manner. The hardware usually provides the basic means to construct a packet, such as the preamble and Start of Frame Delimiter which are automatically added by the physical layer, and provides several signals and methods to start a transmission or reception and keep track of the process. When conceiving new ideas at the lowest level, the devices can be used without the aid of any Operating System (OS), however, several OSs are available that were specifically designed for sensor nodes, such as TinyOS and Contiki, where support is provided for the higher communication layers. Note that since the hardware allows new designs at a low level, both CSMA and TDMA protocols are supported.

A WLAN is well known for its adoption in the office and home environments, where people are working on their laptops, checking the Internet over their wireless connection, reading the latest news or enjoying a movie on their tablets, etc. However, the WLAN technology, nowadays usually based on IEEE Std 802.11, marketed under the Wi-Fi brand name, is also employed in other fields, such as long range communication between ships at sea or in order to offer communication to rural areas in India. Moreover, the technology offers such interesting characteristics that it is also being used for multimedia communication, such as videoconferencing, or wirelessly displaying a presentation. On top of this, thanks to the widespread usage and integration of WiFi compatible chipsets in products, the cost of the chipsets has become reasonably priced, thereby making the technology extra attractive. Note that both the operational cost as well as the procurement cost of WiFi chipsets is a certain factor higher than the cost of sensor devices [3]. WiFi chipsets are more expensive, consume more energy and require a more complex, and therefore more expensive, microprocessor to control the radio chip. However, the technology is developed for an entirely different market segment, where high throughput and moderate power consumption are considered as important, whereas sensor nodes are developed for low throughput and low power consumption. A WiFi chipset is usually supported by a general purpose OS, such as Linux, which provides an entire network stack to facilitate the wireless communication. As such, several modes of operation are supported by the stack, such Access Point (AP), mesh and ad hoc mode. New evolutions in the IEEE Std 802.11 standards result in the continuous extension of the network stack in order to keep up to date with the latest developments. Such developments do not only impact the in software implemented network stack, but also the hardware itself. Certain functional components have been implemented in hardware, such as DCF and automatic acknowledgements, and provisions are made for QoS by providing several queues with adjustable parameters. The latest standards even force the hardware developers to abandon the software MAC functionality, which needs to run on the dedicated radio chip processor in order to obtain the promised performance. As a result, the WiFi technology can not be adjusted at such a low level as sensor nodes, resulting in fewer design options regarding research.

A Cognitive Radio (CR) can be defined as a fully reconfigurable wireless transceiver which automatically adapts its communication parameters to network and user demands [4]. Around the turn of the millennium, researchers started to realize that certain bands of the radio spectrum were not used, either at all or not used all the time, or would become available thanks to the switch from analog to digital broadcast television, which was able to compress that data in a much more efficient manner. Dynamic Spectrum Access (DSA) is a specific field of Cognitive Radio which employs such white spaces, that is, spectrum bands which are unused. The radio identifies the portions of the spectrum that are not being used [5]. In this work the concept of Cognitive Radio Network is limited to the Neighborhood Discovery phase in the case in which an entity, also called the Primary User (PU), which operates in a licensed band does not use the band in its full capacity, allowing opportunistic channel access by Secondary Users (SUs). Since the SUs exploit these frequencies in such manner, they should not interfere with the transmissions of the PU. Such access scheme clearly impedes the communication between SUs. While a communication path has been established by the SUs, the PU might initiate transmissions, causing the SUs to abandon the selected frequency band. SUs therefore require an adaptive and dynamic method to establish a communication link. Rendezvous protocols provide a such a method which allows SUs to meet on specific channels for a minimum duration of time. Because of the extremely dynamic context in which the nodes operate, a flexible hardware platform is required, on which designs can be implemented on a extremely low level, even lower than on sensor nodes. The employed hardware in CRNs consists usually of a Software Defined Radio (SDR), where even parts of the physical layer can be programmed. Examples of such hardware platforms include the USRP and WARP platforms [6] [7].

The remainder of this chapter starts with a discussion of the MAC layer and its position with respect to the other networking layers. Section 1.2 considers the pros and cons regarding contention based and scheduled medium access. The timely scheduling of transmissions or tasks is tightly related to real-time scheduling. Therefore, a limited introduction in real-time systems is provided in Section 1.3. The following section discusses the contributions of this work and the final section concludes this chapter by indicating the relation between previous publications and the chapters of this work.

1.1 Networking stack

In order to send a packet successfully from node A to node B over a network, a whole range of operations need to be passed. Assuming that the destination address

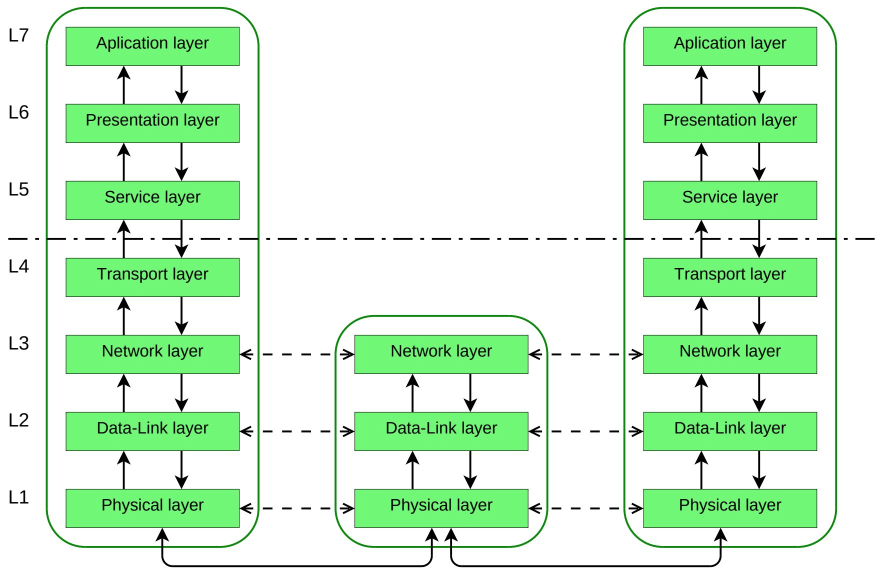


Figure 1.1: *ISO OSI reference model.* *(Ref: Adapted from Computernetwerken, 2e herziene uitgave, A. Tanenbaum [8], Figure 1-16, page 30.)*

is known, a path towards the destination can be found. Such a path consists of a series of intermediate node addresses, or at least the following address, which need to be passed in order to reach the destination. In wired communications, such path depends on the number of routers and subnets the packets needs to pass through. In wireless communications, such path is determined by the physical distance between the two radios and the routing protocols which are implemented in the specific network. Besides the route towards the destination, the nodes need to follow a specific protocol in order to communicate to each other. The transmission towards the following node in the path needs to follow this protocol, in order to make sure that for example the packet has been received by the node, that the transmission does not interfere with other ongoing transmissions, etc. Moreover, too large packets need to be fragmented and are sent in multiple packets over the link. Both wired and wireless technologies define a certain upper bound on the packet size, based on the capabilities and properties of the physical transmission method. Due to retransmissions or different paths, it might happen that packets arrive in a different order at the receiver than in which they were transmitted. The receiving side therefore needs to make sure that all packets are rearranged according to the original order before defragmenting the packets again.

It is clear that a multitude of functional entities needs to be provided by the network, which can easily become confusing, thereby clouding the global picture. In order to be able to explain and understand data communications between different networking technologies, the ISO OSI (Open Systems Interconnection) reference model [9] was created, which is depicted in Figure 1.1. The model is composed out of seven networking layers, each depicting a specific networking function which is realized by a network protocol. Each layer provides the necessary abstractions towards the lower and upper layers. The upper three layers refer to the functionality required when the data is either originating from or destined to the userspace of the Operating System. The upper layers are not discussed here, since they are either related to the specific user application, or the functionality is already provided in contemporary lower layer protocols. The lower four layers address the reception and transmission of data when simplifying their functionalities to the extreme. Those layers present a framework according which protocols can be designed. A protocol stack is formed by a set of protocols which interact with the protocols on the lower and upper layer. Note that the reference model does not dictate the strict separation between the layers. It is known that some protocols transgress the boundaries of the layers. The protocol stacks related to this work include the TCP/IP stack and the ZigBee stack, which both define four layers. The TCP/IP stack defines the host-to-network layer, Internet layer, Transport layer and the Application layer. The ZigBee stack on the other hand defines the Physical layer, Data-Link layer, Network layer and the Application layer. A rudimentary description of the lower four layers of the OSI reference model is provided next.

The lowest layer in the reference model is represented by the physical layer. This layer is considered to be different for each communication technology and medium. It defines the available modulation schemes, the encoding on the medium and therefore implicitly also the maximum transmission rate, the required adapters to connect to the physical interface, whether or not the interface supports full-duplex, etc. The characteristics consider mainly mechanical and electrical properties and the physical medium over which the data transmission is carried. Because of the strong link between the hardware and technology, usually the physical layer is implemented in hardware and is not to be modified. In order to allow the higher layers some control of the physical layer, most hardware allows to configure some physical parameters. An exception to the fixed hardware design is Software Defined Radios (SDR), where the physical layer can be defined and programmed on the hardware.

The data-link layer, also referred to as L2, is the layer which is responsible for peer-to-peer communications, that is, communication between nodes within the same local area segment. Communication to nodes beyond the local LAN segment is to be orchestrated by the higher layers. The L2 layer solely provides the means to communicate to the next node in the path. According to the OSI reference model, the data-link layer is composed out of two sublayers, the Logical Link Control (LLC) and the Medium Access Control (MAC) sublayers, each with its own specific functionality. The LLC sublayer is used for data link layer addressing, flow control, address notification and reliable transmission over the link. The MAC sublayer controls the access to the physical medium, attempting to ensure a collision free transmission when multiple nodes compete for the same physical link. However, in actual protocol implementations, the two sublayers might be difficult to discern. In fact, some protocol stacks, such as ZigBee do not define an LLC sublayer, instead only a MAC layer is defined, which also incorporates the functionalities that are theoretically assigned to the LLC. In this work the MAC is therefore considered as the L2 layer, which is for example responsible for reliable peer-to-peer transmissions, medium access mechanisms, addressing, etc. Reliable transmissions require the means to detect and whenever possible correct errors that may have occurred at the physical layer. In wireless communications, there are plenty of reasons why a transmission may have failed, a too low Signal to Noise Ratio (SNR), too much interference, bad bit synchronization, multipath fading, etc. As a result of those errors, the receiving radio might miss the packet transmission, or might detect some wrong bits, that is, there are some bit errors. In the first case, the packet is not received at all, thereby creating a situation where the receiver is unaware of any failed transmissions and thereby also unable to inform the transmitter of the failure. The L2 layers provides means to recover from such situation, allowing the transmitter to be informed even when the receiver is unaware of the failed transmission. Received packets containing bit errors can be identified as failed transmissions by for example a straightforward Cyclic Redundancy Check (CRC). The L2 layer is capable of informing the transmitter about an erroneous packet reception, which might invoke a retransmission or the receiver might make an attempt to correct the erroneous bits by means of an error correction mechanism, such as Forward Error Correction (FEC). Note that nodes transmitting at the same time can result in failed receptions. Therefore, the medium access control is such a vital functionality of the MAC. The MAC will provide means such that the probability that two nodes are sending at the same time can interfere with each other is minimized, i.e., the MAC attempts to prevent collisions. Upon such collision, specific recovery mechanisms are provided, as a collision is considered as one of the errors that may occur at the physical layer. Note that MAC protocols can be classified according to their channel access method, of which the two most prominent classes are contention based channel access (CSMA/CA - Carrier Sense Multiple Access/Collision Avoidance) and scheduled channel access (TDMA - Time Division Multiplexed Access). The pros and cons of each of the classes is depicted in the following section. For both networks for which this work considers the MAC, WLANs and WSNs, the MAC provides a similar yet different functionality. Both provide control over the medium access, are able to provide QoS, can provide addressing of the nodes and consider power efficiency either by design or by including a power saving protocol. However, the MAC of the WiFi, as defined by the IEEE Std 802.11 standard, incorporates a significant amount of functionalities more than the MAC of the WSN. The former operates in the comfortable context defined within a General Purpose Operating System, whereas the latter needs to be able to operate on resource constrained devices. Specific MAC protocols for both network types are discussed in more detail in the relevant chapters.

Whereas the MAC is responsible for local addressing and providing the peer-to-peer communication, the networking layer is responsible for providing a logical address and the routing between networks. The logical address indicates both the node and the network in which it resides, thereby enabling not only communication with nodes of the same LAN segment, but also with nodes in a different network. A second responsibility includes the routing between networks. In the ZigBee stack [10], this layer is concerned with multi-hop routing.

The transport layer considers a reliable transmission of data. The received data should be free of errors and in the same order as it was transmitted. The layer differentiates between connection-oriented protocols and connectionless protocols. Two of the most know transport layer protocols are TCP and UDP.

1.2 Uncoordinated vs coordinated channel access

This section discusses the pros and cons of two of the most notorious classes of medium access mechanisms, contention based channel access, also known as CSMA, and scheduled channel access, also referred to as TDMA. Most researchers have a certain preference for one of these access mechanisms, which usually coincides with the type of research they are performing. One of the access methods is usually more suited for a specific research area, when compared to the other. This section discusses both mechanisms in an objective manner, without specifying a certain targeted research field.

Carrier Sense Multiple Access (CSMA) is an access mechanism where the nodes are said to be contending for the channel access. CSMA is an uncoordinated access mechanism, that is, there is no negotiation or agreement on the time the channel can be used by the nodes. As a result, the nodes need to decide for themselves whether it is advisable to start a new transmission or wait until the medium becomes free. Therefore, before transmitting data, the nodes first listen to the activity on the channel. When the channel is assumed to be busy, the nodes defer their transmission and either wait until the channel is free again, or wait until they are allowed to make a new attempt to access the channel. The precise methodology depends on how collisions are dealt with or are attempted to be avoided. The CSMA/CD (CSMA with Collision Detection) is used for Ethernet transmissions [11], where the links are determined by the cabling. Since there is a physical wire over which the signal is transfered, a collision can be detected during the transmission. When a node detects that its transmission is colliding with the transmission of another node, it aborts its transmission, sends a jam signal to inform the nodes about the corrupted packet and enters a wait state. During this state, the node performs a random backoff before attempting a retransmission of the data. The waiting time of the random backoff is determined by means of a Binary Exponential Backoff (BEB) mechanism.

Since wireless communication is unable to detect a collision based on the voltage on the wire, a different collision handling is proposed for wireless transmissions. The method is called CSMA/CA (CSMA with collision avoidance), where the nodes also sense for the medium to be idle before attempting a transmission. When the medium is not considered to be busy, the node is allowed to transmit immediately. If an ongoing transmission is detected, the node waits until the end of the transmission, waits for a specific time, determined by the standard, and then performs a random backoff. After the backoff period, the node senses the channel again and attempts a transmission when it is considered to be free. Collisions are not so easily detected, therefore the standard includes an acknowledgement mechanism to inform the nodes of a successful transmission. Upon a failed transmission, the node attempts a new transmission according to the channel access rules. Upon a failed transmission, the Contention Window (CW) is enlarged in a binary exponential manner. The new backoff time is selected randomly from the CW value range, thereby providing a larger probability for nodes to have a different backoff period and thus enhancing the probability of a successful transmission. Interesting about this method is that a certain QoS can be provided by means of specifying a lower CW range for higher priority packets, such as defined in the IEEE Std 802.11n standard [12]. Note that according to the IEEE Std 802.14.5 standard the nodes first perform a random backoff before sensing whether the channel is idle.

One of the major advantages of CSMA is its scalability. It makes no assumptions regarding the size of the network, neither does it require such information. Moreover, thanks to the carrier sensing and the random backoff, the method is robust to interference and is able to adapt to the variable network conditions. Moreover, by adjusting the time before starting the backoff time and the window in which a random value can be selected, a certain QoS can be provided. Whereas the lack of coordination ensures the necessary flexibility and adaptiveness, it also requires neighboring nodes to keep listening for incoming packets. As such, when no data is being transmitted, a lot of power is wasted due to idle listening. Likewise, when data is being received that is not destined for the current node, which is called overhearing, energy is wasted. Note that the power consumption of a reception is in most radio chips comparable to the power consumption of a transmission. Moreover, in dense networks the uncoordinated operation of the access method results in a lower overall throughput, and therefore also a lower bandwidth usage efficiency. When a considerable number of nodes attempt to send at the same time, either the backoff, collisions due to hidden nodes or virtual carrier sensing result in a considerable amount of time not spent for transmissions. Note that although the method allows the possibility for a certain QoS, no guarantees can be provided for a fair bandwidth allocation between the nodes. Moreover, even though QoS can be used, the time required to access the wireless medium is not bounded.

Time Division Multiplexed Access (TDMA) mode on the other hand ensures a coordinated medium access. All nodes operating within the network are assigned a duration of time, also called time slots, during which they are allowed to access the wireless medium. The set of time slots are typically organized in frames, which denote a collection of time slots. Note that the allocation of time slots can be assigned in a periodic manner, not necessarily every frame, but can also be assigned in an on-demand base. Once a node has access to its designated time slot, it performs no channel sensing like CSMA mode dictates, it assumes the channel is free to transmit. The control of the medium access is organized such that no collisions happen between the nodes and therefore, no retransmissions are required. Such behavior is only possible when the network and access method comply to certain conditions. First, the slots should be assigned such that no neighboring transmission will interfere with the current transmission. The phrasing already implies that when nodes are sufficiently geographically separated, such that their transmissions can not collide with one another, not even at a common receiver between both nodes, then they are allowed to use the same time slot. Second, the nodes should become synchronized since the entire slot allocation is relative to a certain reference time. When nodes are out of sync, that is, the time reference of one node is different from the time reference of another node, then nodes start to transmit packets outside of their designated time slot, possibly resulting in collisions with other transmissions. Third, nodes should maintain their synchronization, since due to the small differences in clock source frequency, the reference time clocks tend to drift apart.

Decades of research have contributed to numerous slot allocation methods. Some methods adhere to a centralized scheduling algorithm, where only a single coordinator can be found. Others employ a distributed method, where each node determines its own time slot based on local information. Such local information can be collected by means of information exchange packets, passive listening, geographical position in relation to other nodes, etc. The advantage of the distributed allocation method is the independence on any central coordinator, thereby eliminating any required feedback towards this single point of failure. Moreover, since each node selects its own time slot, there is no problem with any joining or leaving of nodes. Any new slot allocation scheme is selected such that there is no interference with the other nodes. The centralized allocation method would require to update the entire network of nodes to let them know about the new schedule, thereby making sure that all nodes effectively employ the new schedule. The downside of the distributed allocation method can be the overhead of messages, in case of time slot information exchange messages, the required geographical location knowledge of other nodes, which is limited because of the available memory, etc.

In contrast to contention based access methods, a medium access according to TDMA mode should not result in collisions. All time slot assignments are such determined that no collisions can occur. Therefore, operating in TDMA mode saves more energy because of the lack of collisions and therefore, retransmissions. Moreover, the TDMA operation ensures a precise control of the transmission and reception of the nodes such that idle listening and overhearing can be avoided, allowing nodes to enter a low power state, thereby reducing the energy consumption. As such, a TDMA protocol is more suited in applications where the energy consumption is a vital parameter or in dense networks, when compared to contention based medium access modes.

Note that a TDMA protocol requires the nodes to be time synchronized. The overhead of such synchronization protocol is often considered as a downside of a TDMA protocol. However, there are very efficient synchronization protocols, which do not require any substantial message exchange in order to synchronize the nodes, some of which are discussed in Chapter 4. On top of that, one should also consider the message overhead of a contention based protocol when retransmissions are required. Moreover, the synchronization ensures a time relation between the nodes, allowing the gathered data to be compared. Another drawback of a TDMA protocol is that it is more complex than a contention based protocol. A contention based protocol does not need to make any agreements with its neighboring nodes, it can transmit its data based on a local sensing method. Nodes operating according to a TDMA protocol need to agree on a certain time slot assignment. Such agreement can either be reached in a distributed or centralized fashion. A last issue which is often being raised as a downside of TDMA protocols is the rigidness of the protocols, that is, the protocol is not sufficiently flexible to changes in the network. Certainly, due to the required time slot assignments which can not overlap with neighboring nodes, acting upon changes in the network is more complex in comparison to contention based protocols. However, the claim of this disadvantage is due to some badly designed TDMA protocols. Protocols where each node is assigned exactly a single slot, without taking into account the required throughput, would indeed be rigid. At a certain point in time, the pool of available time slots is depleted and new nodes attempting to enter the network are refused. Such protocols determine the maximum number of nodes at design time. Such protocols might work excellent for the task for which they were designed, but offer a poor performance in other environments. An intelligent design, which considers the required throughput of the nodes is not obstructed by this issue, neither is an algorithm which makes use of spatial reuse of the slots.

1.3 Real-Time Systems

A common misconception is that real-time systems means the systems need to be fast, or that when a system is sufficiently fast, it can perform real-time tasks. Nothing could be further from the truth, a real-time system deals with reliability and guaranteed response times within specified deadlines. Even a very slow system, taking hours to complete a single task, could be real-time. This section elaborates on some basic principles of real-time systems and in order to start with it, first a quote of a definition of a real-time system [13] is presented:

> Any system in which the time at which output is produced is significant. This is usually because the input corresponds to some movement in the physical world, and the output has to relate to that same movement. The lag from input time to output time must be sufficiently small for acceptable timeliness.

The definition indicates a certain relation to the time required to perform a specific task. However, note that the time duration should be sufficiently small within the context of the specified system. For example, a missile guidance system will require a faster response time than a word processor which records the keystrokes from the keyboard in order to be considered as real-time. The most known definition of a real-time system includes that *the correctness of a real-time system depends not only on the logical result of the computation, but also on the time at which the results* *are produced* [13]. In other words, every task should be finished within the specified deadline, otherwise the results can not be considered as valid and could even have disastrous results.

However, a distinction needs to be made between the effect of receiving a too late response. When the word processor is not able to produce the characters on time, it is not considered to be a disaster. On the other hand, the too late response of a missile guidance system could have catastrophic consequences. In order to convey the concept of importance of reaching the deadline, a differentiation is made between hard and soft real-time systems. The former requires that every task is executed within the specified deadline. Not reaching the deadline is not acceptable for the system and could cause much harm to either the operation of the system or its environment. Soft real-time systems, on the other hand, consider the execution within the specified time important, but is able to handle an occasional task whose execution was too late [14].

The key element of such real-time system is obviously the scheduling of tasks. Over the years, a considerable collection of scheduling algorithms has been constructed by researchers all over the world. The scheduling algorithms are typically categorized according to the type of task they are trying to schedule. Some algorithms are only applicable to periodic tasks, whereas others specifically aim for aperiodic tasks. Algorithms can also be discerned based on whether the scheduling is dynamic, that is, the scheduler determines at run time the next tasks which should be scheduled, or is a static scheduler. Certain algorithms take into account the dependencies of the tasks, whereas others assume an independent task. This discussion does not go further into the details of these scheduling algorithms since they are out of the scope of this work.

The designer of a real-time scheduler needs to consider only the tasks at hand. Imagine designing a real-time operating system, where multiple tasks need to be performed in the background, one needs to consider whether or not to support preemption, interrupt handling, whether or not priorities are assigned to tasks, etc. Ensuring the real-time behavior in such system becomes a strenuous task. A limited set of factors which have an impact on the real-time behavior of an operating system is discussed here, such as preemptiveness and interrupt handling [15][16].

Preemption is the ability of a kernel to halt a currently running process or task in favor for a higher priority process. In non-preemptive kernels, also called cooperative multitasking, a task needs to explicitly abandon control of the CPU in order to allow another task to run on that CPU. Note that the user believes that all tasks are running concurrently, which implies frequent context switches from one task to another. The major disadvantage of such preemption model is the low responsiveness. All tasks need to wait until the currently running tasks relinquishes the CPU, even the high priority tasks. As a result, the response time is nondeterministic, as it is unknown when the high priority task will acquire control of the CPU.

Preemptive kernels can preempt any task in favor of a higher priority task, thereby increasing the system responsiveness. The activation of a higher priority task triggers a context switch, such that it can execute immediately. As a consequence, the execution of the task with the highest priority is deterministic. Therefore, a considerable number of real-time kernels, such as µC/OS-II, are preemptive kernels[15]. Note that such preemption model requires some extra care to be taken regarding shared data, since any task can be preempted at any time. Moreover, preemption entails some extra overhead due to the storing and retrieval of the CPU registers, the state of the current task, etc. Since preemption is based on the priorities of the tasks, the priority assignment to tasks suddenly becomes a crucial undertaking. The priority of a task can also depend on the type of scheduler which is being used. For example, one of the available methods, Rate Monotonic Scheduling (RMS), considers that tasks which need to be scheduled more frequently, need to be assigned a higher priority. One of the assumptions of the method is that all tasks are considered to be periodic. Issues can also arise specifically due to this priority assignment when used in combination with locking mechanisms. One of such issues is known as priority inversion, where a higher priority task is reduced to a low priority task due to some shared locking. Several resolutions have been presented to prevent or resolve such situations. Further details are out of the scope of this work.

In systems that support Interrupt Service Routines (ISR), irrelevant of which preemption model is used, any task can be interrupted by an asynchronous interrupt handling. When considering the timely and deterministic execution of high priority tasks, such an interrupt might have a significant influence on the real-time behavior of the system. On the one hand, interrupts ensure the immediate processing of an event, but on the other hand, the interrupt processing of a less relevant event could deteriorate the performance, and therefore also the determinism, of a high priority task. Although the opinions vary, a polling based system could ensure a higher determinism compared to an interrupt based system. However, the determinism comes at a cost of extra processing overhead and a lower response time. The polling based system could be combined with a couple of interrupt based notifications, such as interrupts from the timer subsystem, which needs to be as accurate as possible.

From this short introduction in real-time systems and real-time operating systems, it is clear that a lot of consideration is required to design such system. When employing the functionality of such system, it is therefore also important to keep in mind the properties of such system in order not to break the rules.

1.4 Outline and main research contributions

Wireless communication is a broad and fast evolving research area, where different fields are the main focus of contemporary research. Although the research fields are typically focused on a single layer within the network stack of a specific type of network, they have some elements in common. Certain methods or concepts need to be adjusted, since each type of network has different priorities, but some commonality can be found. This works is also mostly restricted to the MAC layer, but in order to show the overlapping properties, this work considers three different network types where wireless communication is used: Wireless Sensor Networks, Wireless Local Area Networks and Cognitive Radio Networks. The work is organized in parts according to the three network types. In order to provide the reader with the necessary background and understanding of the concept that is being discussed, is the first chapter of each part acting as an introduction. As such is Chapter 3 providing relevant information about Wireless Sensor Networks, Chapter 6 depicts background information on the IEEE Std 802.11 standard, the Linux kernel and the implementation of IEEE Std 802.11 within the kernel and real-time operating systems. Chapter 8 unveils generic information regarding Cognitive radio Networks and Neighborhood Discovery issues.

The first part considers both synchronization and MAC protocols within WSNs. Chapter 3, which acts as the introductory chapter in this part, provides an introduction in WSNs and the most notorious MAC protocols and their extensions or improvements in order to give an idea of the research interest regarding MAC protocol development for WSNs. The discussion is targeted towards both contention based and scheduled medium access methods, or a combination of both. The chapter provides some insights in the issues surrounding the academic study of MAC protocols for WSNs, whereas some real-life deployments shed some light on more practical issues. Note that some protocols, which make invalid assumptions, are not considered in the discussion due to their inappropriateness for implementations.

The focus of Chapter 4 lies on synchronization methods, both for distributed systems as for WSNs. Certain general synchronization concepts are discussed, such as the reason why synchronization is necessary when two nodes meet each other for the first time and the reason why resycnhronization is vital in such systems due to certain hardware and physical phenomena. The main approaches for synchronization are discussed within the context of typical sensor network platforms. The discussion clarifies the possible dependency of a synchronization protocol on its platform. Therefore, the relevant details of the hardware are discussed and then applied to the synchronization methods. Based on the analysis of the synchronization protocols in the context of actual hardware a more optimal synchronization method is proposed.

Chapter 5 focuses on a MAC protocol for WSNs which provides a scheduled medium access. The protocol employs an intelligent time slot assignment, based on the required throughput of the sensor node. Even though a centralized scheduler is being employed, the time slot assignment is arranged in such manner that no update of all nodes needs to be performed in the case of network changes. Nodes are free to leave and join the network as they wish, provided there is sufficient bandwidth to support their bandwidth requirements. Interesting about the protocol is that it employs a similar consideration as Rate Monotonic Scheduling (RMS), where the data (tasks) are to be scheduled periodically and the highest rate is assigned the highest priority during the slot allocation. As a result, the latency is deterministic, which allows the protocol to be used in a real-time environment. The chapter provides an extensive analysis on the performance of the protocol with different parameters.

The second part focuses on the non-standardized operation of Wireless Local Area Networks. Chapter 6 provides an introduction on the motivations that lead to the usage of standardized hardware in a non-standardized manner. A short summary of the MAC functionality as described in the standard identifies the provided functionality of most hardware and network stack combinations. Related work indicates the application areas and the methods to employ the commercial hardware in such manner. Since the Linux kernel and its open source character has a significant role in such operations, an introduction is provided in the process scheduling, interrupt handling, timer subsystem and the network subsystem. Since the Linux kernel does not provide any hard real-time guarantees, some kernel extensions which either approach or enable a real-time behavior are discussed in combination with a real-time operating system, which was designed independently of the Linux kernel.

Chapter 7 investigates the possibility of using an IEEE Std 802.11n commercially off-the-shelf available hardware chipset in a non-standardized manner, that is, in a scheduled access manner. First an analysis regarding the performance of different timer sources is made, also in combination with real-time extensions and real-time operating systems. Based on the preliminary results, a timer source has been selected and the relevant Linux networking subsystems have been adjusted to enable the precisely controlled slotted transmission algorithm. Unlike a considerable number of works, this chapter considers the presence of foreign entities, resulting in severe interference and obstructions for timely transmissions.

The last part considers Cognitive Radio Networks and more specifically Rendezvous protocols. Whereas the previous parts were oriented towards the development and operation of MAC protocols, this part is targeted towards a stage which is required even before the MAC protocol comes into play. Chapter 8 provides a general introduction in the matter of Rendezvous protocols and discusses the different methods that can be employed.

Chapter 9 discusses the principle of Rendezvous in the context of an asynchronous environment. Three different methods are proposed of which each makes use of asynchronism. The first method depicts an approach to analyze Rendezvous protocols in an asynchronous manner, showing interesting results. Based on the outcome of the first analysis, a protocol extension is designed which ensures asynchronism for any Rendezvous protocol. As a result, each protocol can be enhanced regarding the Time-To-Rendezvous and rendezvous opportunities by applying this extension. Note that the performance analysis is not based on only the first possible encounter between two nodes, but is a statistical evaluation of all possible rendezvous. The third method proposes an improvement of the second method, thereby providing a better distribution of the slot size between the high priority and low priority channels. In combination with a protocol specifically designed to operate in combination with the improved extension, it outperforms all other Rendezvous protocols significantly.

1.5 Contributions

This work provides a contribution in several areas. The first contribution is the analysis of the hardware of two platforms in terms of timing dependencies. Based on those findings, the related work was evaluated and a new synchronization method is proposed, which provides an abstraction of the concept timestamp. The second contribution is a novel TDMA protocol that ensures a reliable operation in a heterogeneous network, while making sure that nodes with small bandwidth requirements still get regularly access to the medium. The protocol results in a deterministic behavior in terms of buffer size and latency. A third contribution comprises the analysis of available timer sources in the Linux kernel or Atheros NICs and the performance analysis thereof in different circumstances. The results have been used to develop a slotted transmission mode with COTS IEEE Std 802.11n hardware, which is another contribution, and describes which issues one can encounter when making an attempt to operate COTS IEEE Std 802.11n hardware in TDMA mode. The final contribution of this work can be found in the area of Cognitive Radio Networks and Neighborhood Discovery protocols, where it was found that making an attempt to a Rendezvous asynchronously offers a higher performance than otherwise. By proposing an extension that induces this asynchronism, every protocol can benefit from this.

1.6 Publications

Parts of this thesis can be found in the following publications:

- part of **Chapter 4** published as a workshop paper, *"WBAN implementation on Magnetic Induction Radio IC for medical remote monitoring", in proceedings of the 1st International Workshop on Medical Applications Networking, Dresden, Germany, 2009.*

- part of **Chapter 5** published as a conference paper, *"QoS support for a MAC with a TDMA tree topology on the Magnetic Induction Radio IC", in proceedings of the 29th Real-Time Systems Symposium (RTSS 2008), Work in Progress session, Barcelona, Spain, 2008.*

- part of **Chapter 5** published as a journal paper, *"Analysis of TDMA scheduling by means of Egyptian Fractions for real-time WSNs", in EURASIP Journal on Wireless Communications and Networking 2011.*

- part of **Chapter 5** published as a conference paper, *"Binary TDMA Schedule by Means of Egyptian Fractions for Real-time WSNs on TMotes", in proceedings of Med-Hoc-Net 2010, Juan-les-Pins, France, 2010.*

- part of **Chapter 7** published as a journal paper, *"TDMA on commercial of-the-shelf hardware: fact and fiction revealed", in International Journal of Electronics and Communications, Vol 69, No. 5, 2015.*

- part of **Section 9.2** published as a conference paper, *"An analysis of asynchronism of a neighborhood discovery protocol for cognitive radio networks", in proceedings of IEEE PIMRC, London, UK, 2013.*

- part of **Chapter 9** published as a conference paper, *"Benefiting from an Induced Asynchronism in Neighborhood Discovery in opportunistic Cognitive Wireless Networks", in proceedings of ACM MobiWac, Barcelona, Spain, 2013.*

- part of **Chapter 9** published as a conference paper, *"AND: Asynchronous Neighborhood Discovery protocols for opportunistic Cognitive Wireless Networks", in proceedings of IEEE WCNC, Turkey, 2014.*

- part of **Chapter 9** published as a journal paper, *"How to be an efficient Asynchronous Neighbourhood Discovery protocol in opportunistic Cognitive Wireless Networks", in International Journal of Ad-hoc and Ubiquitous Computing, Vol 20, No. 3, 2015.*

CHAPTER 2

Research Questions

This chapter enumerates the addressed research questions and the approach that was used to tackle these issues. This work concentrates on a number of research questions in different Wireless Networks. One major theme that is present in all the issues, concerns the consideration of the targeted network in real-life. All research in this work takes into account the hardware on which the protocol should be run or considers the environment and conditions in which the protocol should run as accurately as possible. Sometimes an issue that needs to be tackled is not limited to a single type of network. Therefore, the research can focus on several Wireless Networks for a single research issue. This works is segregated in three parts, each part containing a chapter on the State-of-the-Art and relevant background information in order to better understand the issues that are present in this research area. Investigating in which context the research is positioned and possible alternative methods in order to resolve the issue is one of the first tasks that needs to be performed. The second issue that is being discussed is the synchronization in a Wireless Network. Most of the work in this work is focused on Time Division Multiple Access, hence the synchronization between nodes is an important issue. Essential for this synchronization is also the timely transmission of packets, which can therefore be considered as a subsection of the synchronization. Since a considerable part of this work focuses on TDMA protocols, where time slots are defined during which data can be sent, the handling of heterogeneousness is an important issue that needs to addressed. Interestingly, the related work is often neglecting this possibility and considers only a single slot size for a specific data type. The related work that considers heterogeneousness often makes use of flexible slot sizes or even dynamic frame sizes. Such approach provides a solution for the problem at hand, however, needs to be considered with care, since it could cause significant issues with regards to synchronization. One final issue that is tackled in this work, is the accurate performance analysis of Neighborhood Discovery protocols in Cognitive Radio Networks. A considerable number of works propose new Neighborhood Discovery protocols, but their performance is often not analyzed in detail. For this reason, this work considers several Neighborhood Discovery protocols, add some enhancements to certain protocols in collaboration with the RWTH University, Aachen, and analyze the performance in an asynchronous network simulation.

2.1 Background

2.1.1 Wireless Sensor Networks

The related work in Wireless Sensor Networks is quite extensive. Whereas this work only focuses on TDMA protocols, it remains important to consider other types of medium access methods, possibly even hybrid methods. It is interesting to note that depending on the targeted application, different approaches have been presented, each of which achieving their goal by means of sometimes unique methods. This work lists the most renowned CSMA protocols, that is, protocols that are often referred to. As regards TDMA protocols and hybrid protocols, this work enumerates protocols that either provide some unique feature that clearly differentiates them from other protocols, or protocols that have a certain feature which might interesting to add to the proposed protocol in this work, in order to improve its performance. The resulting list of protocols forms the content of Chapter 3

2.1.2 Wireless Local Area Networks

The context of the second part of this work is somewhat different, since this part intends to use commodity hardware in order to improve the performance of a Wireless Local Area Network. The work started with a request from one specific party, whether or not a precisely controlled TDMA protocol would be possible on commodity IEEE Std. 802.11n hardware. This soon led to the question whether there was other research that was tackling this very same issue. The resulting list of research papers indicating the need to adjust the standard operation of the IEEE Std. 802.11n commodity hardware is provided as an introduction in Chapter 6. Since it is quite a complex matter to adjust the operation of such commodity hardware, a thorough investigation of the IEEE Std. 802.11n standard is required, listing the functionality which could influence the precision of a TDMA protocol on hardware that makes use of the standard. The most critical functionality is listed as a result in Chapter 6. Important is also the question in which degree the operation of the commodity hardware can be controlled by means of modifying the Linux kernel internals. This requires a thorough understanding of the Linux kernel subsystems, of which the most relevant sections are discussed in the relevant chapter. Since TDMA requires a precise control of the transmission and reception timing, real-time extensions and real-time systems are also considered, which are also shortly described in Chapter 6.

2.1.3 Cognitive Radio Networks

When performing Neighborhood Discovery in Cognitive Radio Networks, it is essential to understand the specific conditions in which this type of network is supposed to operate. The background of this type of network is therefore discussed in Chapter 8. Mathematics provides interesting concepts regarding convergence, which makes it worthwhile to investigate those concepts and a couple of those convergence theories are therefore listed in the respective chapter. Some of the related work, which considers Neighborhood Discovery in Cognitive Radio Networks is also discussed in this chapter, in order to understand the context in which these protocols need to operate and to be able to compare whether the newly proposed protocol can be considered an improvement to the State-of-the-Art. Interesting is also the manner of evaluation whether one protocol is superior to an other. This led to the conclusion that the protocol evaluation is often insufficient, since asynchronous situations are often disregarded, whereas this constitutes the natural state of such network.

2.2 Synchronization in Wireless Networks

2.2.1 Wireless Sensor Networks

This work focuses on TDMA protocols in Wireless Sensor Networks, where synchronization between the nodes is essential for a reliable operation of the protocol. This research addresses multiple questions regarding synchronization, such as "Why is synchronization required, what are the physical processes that influence the synchronization precision?", "Which synchronization protocols are available and what is their performance?", "Is there a more efficient method to ensure synchronization?". These research questions are addressed in Chapter 4. In order to understand the physical processes of time drift, the hardware schematic of a typical sensor node has been analyzed, as well as the internals of the clock related subsystems of the microcontroller on such node. The result of this analysis of these physical processes is used as an introduction to the synchronization chapter. In order to have an idea of which types of synchronization methods are available for Wireless Sensor Networks, this work first focuses on a general synchronization theory, in order to obtain knowledge about the theory regarding synchronization. Then an overview of existing Sensor Network Synchronization protocols is made and organized according to their employed methodology. In order to be able to compare the protocols methodologies, their approach has been applied to a theoretical model of a specific microprocessor, which allows to analyze and more importantly, compare the performance of such protocols. This led automatically to the question whether or not there exists a more efficient method in order to synchronize the nodes in a network. By eliminating specific dependencies on the physical duration of processes, a protocol was designed which resulted in less overhead, a lower protocol complexity, while at the same time having a small dependency on the employed hardware.

2.2.2 Wireless Local Area Networks

The synchronization between nodes can be easily achieved in WLANs, since the hardware has a built-in timestamping mechanism in for example AP mode, to which the stations can synchronize to. The frame synchronization is therefore accomplished by the hardware functionality. Slot synchronization on the other hand, is non-existent in such networks. Since the fundamental operation of the standard hardware is CSMA mode, the question needs to be raised to which degree it is possible to modify the behavior of the hardware and to which degree changes need to be incorporated in the Linux kernel. Moreover, in order to obtain a reliable transmission scheme, the question needs to be raised whether there are timers available in either hardware, the Linux kernel, potentially with a real-time extension, or a real-time Operating System which provide a sufficiently precise timer interrupt to serve the packet transmission scheduler. Chapter 6 refers to the search for background information, which was required to understand in detail how the Linux kernel operates, which processes, system interrupts, etc could influence the reliable operation of a timer source, how the kernel is controlling the wireless network card and determine which actions are taken by which kernel modules. Moreover, information needed to be gathered how the hardware clock input is able to generate a software clock source and determine how accurate this operation is. Part of the result of this information gathering is used as background information in Chapter 6, and this information is also employed to understand and implement the required changes in the Linux kernel. Moreover, the operational knowledge of the kernel and WLAN hardware, allowed to execute a performance analysis on several timer sources, in order to obtain a measure of how precise those timer sources could be. In order to verify whether the system would exhibit a similar performance when the system is overloaded, the timer analysis was performed in different conditions, with a number of stress tests that could influence the timer precision. The result is shown in Chapter 7, explaining the timer analysis in detail. During this investigation, the question was also raised whether or not the entire set of timer interrupts of the system would not result in an overloaded timer system. In order to verify this, a real-time timer subsystem was created on an ARM platform where the real-time timer subsystem received its clock input from a second hardware timer. The results showed that both timers were able to show a performance that was comparable to each other.

All previous research questions and results were the enabling factor to adjust the Wireless LAN driver such that certain hardware functions were disabled, which led to a less CSMA mode. On the other hand, they provided the necessary information to adjust the network drivers such that a precise timer source could trigger the packet transmission. The resulting packet transmission was very accurate, exhibiting only a small number of packets where the deviation from the intended transmission interval exceeded 10 µs.

2.3 Heterogeneous Wireless Networks

An question that was left unanswered by most of the Wireless Sensor Network State-of-the-Art, is how to ensure an efficient TDMA protocol, having a fixed time frame, fixed time slots, operating in a heterogeneous network, where sensor nodes might require a bandwidth that is multiple times higher than the bandwidth demand of other nodes. Another question that influenced this work, was how to ensure the network is able to operate in a stable manner, even while nodes are joining and leaving the network, causing disruptions in the network connectivity, transmission scheme updates, etc. In other words, how to ensure a robust network which is able to operate in an autonomous manner. The last question that triggered this research, was how to provide fairness in such network, such that high bandwidth nodes would not occupy the wireless medium at the expense of the low bandwidth nodes.

Interesting is to note that a lot of work in the previous century was targeted at providing fairness in wired networks. An extract of the most interesting parts of this related work is shown in Chapter 5. Note that real-time scheduling problems also attempt to find a schedule in which tasks can be scheduled such that the total set of tasks is guaranteed to finish in time. However, since there is a huge conceptual difference between executing a task and sending a packet, no reference is made towards real-time scheduling, although it surely provides interesting background information. Real-time MAC protocol on the other hand are described Chapter 5, since the proposed TDMA protocol exhibits real-time characteristics with regard to latency determination.

The research in Chapter 5 focuses on a MAC protocol for WSNs which provides a scheduled medium access that complies to all these demands. The protocol employs an intelligent time slot assignment, based on the required throughput of the sensor node. Even though a centralized scheduler is being employed, the time slot assignment is arranged in such manner that no update of all nodes needs to be performed in the case of network changes. Nodes are free to leave and join the network as they wish, provided there is sufficient bandwidth to support their bandwidth requirements. Interesting about the protocol is that it employs a similar consideration as Rate Monotonic Scheduling (RMS), where the data (tasks) are to be scheduled periodically and the highest rate is assigned the highest priority during the slot allocation. As a result, the latency is deterministic, which allows the protocol to be used in a real-time environment. The chapter provides an extensive analysis on the performance of the protocol with different parameters.

2.4 Neighborhood Discovery performance

In Cognitive Radio Networks, some Neighborhood Discovery protocols make great efforts to synchronize the nodes, before being able to exchange neighborhood information. However, it has been shown that in some cases asynchronism is beneficial towards the performance of such protocol. However, firstly, the question is in which situations is this asynchronism beneficial and when will it result in a decrease of the performance. Secondly, the nodes require a certain amount of overlap between their slots. How much overlap is required in order to ensure that nodes can still communicate to each other? Thirdly, if asynchronism ensures an improved performance, are there ways in order to ensure the protocol is operating in an asynchronous manner? As first step towards the asynchronism in Neighborhood Discovery protocols, a methodology was designed that enabled the evaluation of asynchronism and its influence on the metrics in Neighborhood Discovery protocols. By evaluating several protocol in such manner, the influence of the amount of overlap between the slots could be measured. Thanks to the gathered information regarding the performance in an asynchronous environment, this work proposes a protocol extension, which ensures asynchronism for any Rendezvous protocol. As a result, each protocol can be enhanced regarding the Time-To-Rendezvous and rendezvous opportunities by applying this extension. Note that the performance analysis is not based on only the first possible encounter between two nodes, but is a statistical evaluation of all possible rendezvous. After analysis of these results, an improvement is proposed in this work, which provides a better distribution of the slot size between the high priority and low priority channels. In combination with a protocol specifically designed to operate in combination with the improved extension, it outperforms all other Rendezvous protocols significantly.

Part I

Wireless Sensor Networks

CHAPTER 3

Introduction in Wireless Sensor Networks

The concept of Wireless Sensor Networks (WSNs) has become increasingly popular during the last decades. Their popularity comes forth from their small form factor, low cost and while they individually portray a limited added value, when organized in a network, they can provide a tremendous amount of information regarding the phenomenon of interest. Due to the small form factor and low cost, the devices are restricted with regard to memory, CPU power and energy. A sensor network usually consists of a large number of these resource constrained sensor devices, that are able to detect one or more environment variables. Often the deployment regions of sensors can be considered as hazardous environments, or not easily accessible places, thereby making the replacement of batteries a difficult, if not impossible, task. It is therefore expected that sensor nodes are able to function in an autonomous fashion during their entire lifetime, which can be expected to be in the range of several months or even years.

It is no surprise that the reduction of energy consumption has become a frequently discussed topic in research targeting sensor networks, considering their energy constraints. In order to preserve the energy, the CPU can be placed in a sleep state, where it consumes less power. Since the power consumption of the radio transceiver is of such considerable extent compared to the required power for the CPU to operate, additional energy savings can be achieved by eliminating unnecessary radio operations which result in a waste of energy. It is well known that listening for incoming packets while no transmissions are expected, is one of the major sources of energy waste. This phenomenon is also referred to as idle listening. The other sources of excessive power consumption include overhearing, collisions, and control packet overhead [17]. Overhearing is the case where a node is receiving data, i.e., the node is in the receive state, but this node is not the destination node of the packet and therefore needs to discard the packet although it did consume energy to receive the packet. Like overhearing, the energy consumption of a transmitted packet that experiences a collision is considered a waste of energy due to the futile operation. Evidently, control package overhead does not attribute to the data transfer and is therefore also considered as a waste which should be minimized.

This chapter provides an overview of methods that existing MAC protocols for Wireless Sensor Networks employ for resolving the known issues. Since the applicability of WSNs, and therefore also the targeted research areas, is extensive, this chapter focuses on a limited set of MAC protocols. It is believed that this set includes the necessary work to provide a general overview of Wireless Sensor Network MAC protocols. Protocols specifically related to the work in the following chapters are discussed in the respective chapters.

The approach at the MAC layer to cope with the energy constrained operation of the sensor is widely varying. A basic differentiation can be made between contention based MAC protocols and scheduled MAC protocols, although hybrid solutions exist. Such solutions use some combination of both types of access methods, where for example the transmission is scheduled but the nodes are still able to contend for slot access. Contention based MAC protocols for WSNs either enhance their channel sensing methods, or define a certain periodic operation during which the nodes are active for only a specific portion of the time. The remainder of the cycle, the nodes are in low power mode. Such approach is also referred to as a duty cycle method. Several versions of such methods and optimizations are conceived to achieve the most optimal performance during specific circumstances, which will be shown in the following section.

A scheduled medium access method, that is, a Time Division Multiple Access (TDMA) or Frequency Division Multiple Access (FDMA) method, or a combination of both, require a precise time synchronization in order to allow a detailed control of the medium access. Whereas contention based algorithms usually do not require such time synchronization, they are also not able to exert such precise medium access control. Thanks to the accuracy in scheduled access, the idle listening is automatically reduced to a minimum, since nodes only listen to the medium when they are the destination of a scheduled neighboring transmission. A wide range of approaches are used to define an efficient slot assignment, such as a distributed or centralized slot assignment, an assignment based on geographical location, cluster based allocation, etc. The disadvantage of such scheduling is the extra overhead incurred by the synchronization protocol. Several such protocols that are common for WSNs are discussed in more detail in Chapter 4.

It is noted that although the network topology is not typically part of the MAC layer functionality, in WSNs the MAC layer takes it in some cases into account as the boundaries between network and MAC layer can be vague. Some protocols make no assumptions at all regarding the network topology, which does not pose constraints on the applicability. However it requires some sort of routing protocol to run on top of the MAC layer in order to ensure the delivery of packets, resulting in a more complex network stack. Protocols which provide a precise definition regarding the network topology, often employ a tree structure, that is, a many-to-one routing tree, since typically all data is forwarded to a single sink. Such tree structure provides a straightforward structure to forward data to the sink by means of a parent-child relationship. All data is forwarded to the parent and eventually arrives at the sink. An often encountered protocol design issue, which often presents itself when organizing the network in a tree while using a scheduled medium access method, is the buffer overflow of parent nodes. Nodes accept packets from multiple children, but often only a single slot is foreseen to forward data to its own parent, which needs to be shared between its own data and the combined data of its children. Solutions exists in the form of aggregated messages, where a single message is considered to contain all data, both from the children as the data from the node itself. The required level of aggregation depends on the size of the data messages and the correlation between the data. Some data can be represented by means of statistical values, such as minimum, maximum and average values, however, for some application domains such approach is insufficient. Some network applications could also allow for processing locally the data and only forwarding the result or even discard the data for its irrelevant contents [18]. An estimation on the viability of aggregation is discussed in Chapter 5. Another solution is to ensure sufficient allocated slots to allow the forwarding of both children data and locally gathered data. The latter solution is called the raw forwarding of data, which provides the advantage of data processing outside of the sensor network. One of the advantages is the elimination of the resource constrained operation from the sensor network, thereby enabling more complex operations, such as floating point calculations. Moreover, by having obtained all data in its raw form, it allows for the easy adaptation of the processing algorithm, without requiring any modification to the sensor network itself. It is noted that if the sensor system supports modularity, such that part of the code can be updated over the air, the processing of an aggregation based data forwarding would also be capable of updating the processing algorithm without requiring all nodes to be re-flashed.

Besides the differences in access mode and technology, protocols can also be differentiated based on their data collection methodology, which depends on the type of application. Some applications do not expect any data to be transmitted, except during an emergency situation, in which case the sensor network should react immediately. Other applications expect to receive, either in a continuous transmission flow or in bursts, a collection of data representing periodic measurements. Sensor networks that support such applications are designed to monitor the environment and typically transmit at regular intervals the collected data. Instead of immediately forwarding the data towards the sink node, certain MAC protocols store the collected data in persistent memory and transmit their data in a burst to the sink when the opportunity arises. Such operation can be employed for example in the gathering of information related to a flock of animals that regularly visit certain fixed areas, such as a pond. An intelligent manner to reduce the required throughput is by means of filtering the collected data and only forwarding the requested data. In order to make such system flexible, it is controlled by a query system, that is, the application sends queries to the sink, which distributes the query to the respective network area that should respond to the query.

As the applications of the sensor network vary, the targeted optimization within the research also differs. The goal usually includes reduction of energy consumption, but can also consider the latency or throughput optimization. The latter is particularly important in networks where the traditional sensor is replaced by a video camera, which generates a considerable amount of data.

The remainder of this chapter discusses an overview of a small set of existing MAC protocols for WSNs, classified mainly according to their medium access method and type of slot assignment. Some interesting MAC protocols, that can be either contention or scheduled based, are grouped according to their data forwarding method. A last section depicts some actual sensor network deployments, where issues that might arise in a realistic environment are discussed. Note that this chapter does not provide an entire survey of existing MAC protocols for WSNs. The discussed protocols provide an idea of possible MAC protocols, interesting and/or similar ideas as the concepts used to design the MAC protocol which is described in Chapter 5, show an alternative way of achieving similar goals or take similar approaches. For a survey of existing MAC protocols for WSNs, can be referred to [18], [19], [20], [21] and others.

3.1 Contention based data transmissions

Contention based MAC protocols are usually proposed for networks that do not have any predefined topology, thereby allowing a flexible MAC operation. The main goal of such protocols is the reduction of energy consumption, and mainly the energy consumption due to idle listening. In traditional CSMA systems, for example in ad hoc networks, all nodes need to keep listening to the wireless medium when not transmitting, according to [22]. This results in a significant energy waste due to idle listening. The protocols in this section alleviate the issue by either introducing a periodic sleep cycle or by means of Low Power Listening (LPL). Both concepts are introduced in the discussed work, as well as work which proposes improvements on the original MAC protocols. Note that the IEEE Std 802.11 standard defines Power Saving Mode (PSM), which allows nodes to enter a sleep state without missing any packet receptions by introducing an ATIM Window during which nodes that were asleep can be informed about buffered packets. However, according to [23] and [24], PSM is insufficient to eliminate the idle listening entirely. When stations have only a moderate amount of traffic to send or receive, is is shown that the stations spend 80% of their time to idle listening, even with PSM enabled. Moreover, while the overhead of the message exchanges might seem negligible in WLANs, it could be considered as a huge overhead in WSNs, where data packets sometimes merely consist of some bytes.

A frequently referred to work on MAC protocol design in Wireless Sensor Networks (WSNs) is called S-MAC [22]. The protocol is designed such that energy efficiency is optimized while still taking into account the scalability of the network. The work identifies several sources of energy wastage, such as packet collisions and thereby associated retransmissions, overhearing, control packet overhead and idle listening. The protocol makes an attempt to reduce the energy wastage from all mentioned sources by introducing three different methods. First, a periodic listen and sleep scheme is introduced. During the sleep period the transceiver is placed in power down or low power mode, whereas during the listen period the node is actively listening to the medium. Second, the protocol makes use of CSMA/CA, not unlike the method defined in IEEE Std 802.11. By employing both virtual and physical carrier sensing, collisions are avoided and nodes which do not participate in a packet exchange are allowed to sleep.The virtual carrier sensing is implemented by means of a duration field in the packets and a Network Allocation Vector (NAV) which maintains the longest known duration. As long as the NAV does not indicate a free medium, the node is not allowed to transmit, even when the physical carrier sensing assumes the medium is free. A third method is the fragmentation of large packets into smaller packets that are sent in a single burst, where for each individual package an ACK message is expected. According to the authors, such method provides more time to access the medium to nodes with more data to sent. It is noted that the proposed medium access mechanism has similar issues as the IEEE Std 802.11 standard. One of those issues is the hidden node problem, where interfering transmitters are unable to sense the other transmitter. In order to avert the hidden node issue, a method similar to the RTS/CTS message exchange is exploited. For the burst transmissions of fragmented packets, only a single RTS/CTS exchange is used, thereby reducing the overhead. Even though the access mechanism is contention based, a certain synchronization is required to align the sleep phase and active phase time intervals. Otherwise nodes would be sleeping and unable to receive any incoming transmissions of neighboring nodes that are in the active phase. In order to reduce the overhead of a synchronization mechanism, nodes are synchronized in a very coarse manner, that is, the clock drift time is considered to be insignificant compared to the listen duration. All nodes periodically broadcast their schedule, that is, the duration until the next sleep time, relative to the transmission time of the sender. The receiver of such synchronization message assumes the transmit time is equal to the receive time and adjusts its counter. Nodes that did not obtain a schedule yet adopt the received scheme, whereas nodes that already follow a scheme, adopt both schemes. Such synchronization method makes certain assumptions which result in an inaccurate synchronization, which is explained in detail in the next chapter.

S-MAC already shows a reduction in energy consumption when compared to conventional contention based protocols. However, due to its fixed duty cycle, that is, its listening period is fixed for the whole network, significant power savings can yet be made. According to [25], all nodes will attempt a transmission at the start of the active period, a large idle period might be present where the radio is still operating, resulting in a waste of energy. T-MAC [25] addresses this issue by defining a dynamic active period, depending on the amount of traffic. The protocol employs similar mechanisms as S-MAC, such as, it employs a comparable synchronization method, the same RTS/CTS data ACK scheme, and a similarly defined active and sleep cycle. However, T-MAC proposes an optimization for the active period duration, which involves prematurely termination of the active phase when no more data is presented. If no communication is detected during a certain timeout interval, the nodes enter a low power mode and deactivate their transceivers. Since, like S-MAC, T-MAC performs synchronization in so-called virtual clusters, boundary nodes might need to provide support for two active periods. Although the shortening of the active period based on the traffic demand effectively reduces the energy consumption, it might also lead to an increased latency when the activity period was terminated too soon. In order to prevent such case, T-MAC proposes a Future Request-to-Send (FRTS) packet, notifying the listening nodes that a packet is being prepared for transmission.

Another protocol where the active period is adjusted to the amount of traffic is TA-MAC [26]. The protocol performs a dynamic adjustment of the duty cycle based on the traffic load, the energy level and the status of the TX and RX buffers. The foundations of the protocol are based on those of U-MAC, a protocol which is not included into this discussion. U-MAC makes adjustments to the duty cycle based solely on the traffic load. During each synchronization period, the nodes determine their utilization factor and adjust their duty cycle accordingly. An issue would arise if a node receives packets from multiple nodes and needs to forward those to multiple nodes. The forwarding node would have a larger duty cycle in order to cope with the amount of traffic passing through. However, the receiving nodes have a smaller duty cycle, which could amount to a situation where the forwarding node makes an attempt to send packets, whereas the receivers are sleeping. TA-MAC makes improvements towards the duty cycle adjustment, such that not only the traffic load is taken into account, but also the remaining energy level to determine a valid duty cycle. Moreover, when the TX buffer is deemed to overflow and the receiver has a lower duty cycle compared to the sender, a packet is sent during the next synchronization cycle to request the adaptation of the duty cycle of the receiver, such that both sender and receiver use an equal duty cycle. The receiver is free to accept or reject the request based on its remaining energy level.

Whereas S-MAC and T-MAC provide not only a link protocol, but also services such as synchronization, B-MAC is focused specifically to the link layer [27]. The differentiation between the protocols is not solely based on the type of service they provide. B-MAC examines the channel sensing mechanism with greater detail and makes use of it in both its collision avoidance mechanism and low power communication. An optional automatic ACK functionality is provided, which can be turned on or off by the configuration methods provided to higher layers, which are responsible for the routing, synchronization, etc. Key to the design of B-MAC is the CCA mechanism and noise floor determination. In order to determine whether the medium is idle, protocols usually collect a single received signal strength sample and compare this to the noise floor. However, such practice leads to a large number of false negatives, that is, the channel is too often deemed to be busy. As a remedy, B-MAC collects a number of samples and attempts to detect outliers, such that the channel energy is significantly below the noise floor. Since the noise exhibits a significant variance in channel energy, while the channel energy of a received packet maintains fairly constant, the only case in which a signal strength below the noise floor can be detected is while the medium is idle. When in one of the samples no such outliers is detected, the channel is considered to be clear. For this Clear Channel Assessment to work properly, an accurate noise floor estimation needs to be made. Since the ambient noise changes depending on the environment, software automatic gain control is used to estimate the noise floor. During a period in which the medium is assumed to be idle, such as after the transmission of a packet, signal strength samples are gathered and placed in a FIFO queue. The median of the queue is added to the weighted moving average with a decay of α to obtain an estimation of the noise floor.

This CCA mechanism is used in B-MAC both for collision avoidance and low power communication. The former is achieved by starting a CCA when a request to send a packet is received and to perform backoff if necessary. The low power communication is realized by means of Low Power Listening (LPL). When a node wakes up, it senses the medium shortly by means of the CCA mechanism and determines whether a transmission is ongoing. If no transmission is sensed the node returns to its sleep state, otherwise the node remains awake to receive the transmitted packet. Since the nodes are not synchronized, they each sense the medium according to their own clock. In order to make sure the data payload is received correctly, B-MAC employs a rather large transmission preamble, which is in direct relation to the wakeup cycle of the listening nodes.

It can be said that LPL facilitates significant energy savings when compared to (idle) listening to the channel. However, the energy savings at the receiver side come at a penance at the transmitter side. A longer duty cycle allows nodes to save more energy, however if a significant amount of data needs to be transmitted, it could be more efficient to provide shorter duty cycles, thereby balancing the energy consumption between the receivers and transmitters. Therefore the system is sensitive to traffic flow changes.

The work in [28] proposes SCP-MAC, a MAC protocol based on scheduled channel polling, such as LPL. In order to alleviate the known issues that are inherent to LPL, the long preambles are eliminated and is thereby not sensitive to variable traffic loads. By introducing a coarse wakeup cycle synchronization method, such as the one employed in S-MAC [22], all nodes enter the active state at the same time. As such, only a short preamble is required to detect network activity. Such short preambles makes SCP-MAC more robust to changing traffic conditions. It needs to be noted that the length of the preamble is dependent on the precision with which the nodes are synchronized. A synchronization ensures a compensation for the clock drift, but the longer the synchronization period, the more the clocks drift apart. Therefore, the preamble needs to take into account a certain guard time, allowing nodes to detect an ongoing transmission, even when the nodes are not in sync anymore. A very small synchronization period would ensure a smaller preamble, however, the synchronization does incur an extra cost in terms of energy since control messages need to be exchanged. Therefore, the work considers an analysis for regarding the optimal synchronization period and channel polling duration such that the energy consumption is minimized. It is noted that due to the synchronized wakeup periods extra care needs to be taken with regard to collisions. SCP-MAC splits up the transmission of the preamble and the data by introducing two backoff periods with two different Contention Windows (CW). During the first backoff period, the transmitter waits for a time randomly selected from CW1, after which it senses whether the medium is idle. If the medium is considered to be free, the transmitter sends its wakeup tone, that is, the preamble. If the medium is sensed to be busy, such as by detecting the preamble transmission of other transmitters, the transmitter defers its attempt to transmit until the next transmission opportunity. This reduces the number of transmitters that need to contend for the channel during the actual data transmission, which is preceded by a backoff time randomly selected from CW2.

Whereas SCP-MAC makes an attempt to reduce the required preamble lengths by synchronizing the wakeup cycles of the nodes, it also introduces an increased contention, overhearing and delay. The work in [29] proposes a protocol, named Asynchronous Scheduled MAC protocol (AS-MAC), which employs duty cycling and Low Power Listening to minimize the periodic wakeup time, like SCP-MAC. However, instead of synchronizing the wakeup time of all nodes in order to shorten the required transmission preamble, nodes store the wakeup schedules of their neighbors. Nodes can either be in the '*initialization*' or in the periodic '*listen and sleep*' phase. During the latter phase, nodes perform LPL every wakeup interval and send a '*Hello*' packet every *Hello interval*, which is used to notify its neighbors of local parameters, such as the wakeup interval, the *Hello interval* and the offset of the periodic wakeup. The nodes gather '*Hello*' packets for a certain time during the initialization phase. The collection of packets needs to be longer than the *Hello interval* in order to collect a complete overview of the available schedules in the neighborhood. Afterwards, nodes determine their own periodic wakeup offset by either a random selection algorithm, or by marking the middle of the longest interval between all offsets of the nodes within the neighborhood. In order to notify all neighbors of the new schedule, the node sends its new schedule to each of its neighbors at their respective wakeup time. After the initialization phase, nodes enter the '*periodic listen and sleep*' phase. During each wakeup interval, a node sends a '*Hello*' packet before performing LPL when the *Hello interval* corresponds to the wakeup interval. In all other cases, the node performs LPL immediately. When the medium is considered to be idle after a specific time, the node returns to sleep state, otherwise the node receives the packet and enters a sleep state afterwards. The senders do not wakeup until the destination is scheduled to wake up. In order to prevent collisions between potential senders, each performs a collision avoidance backoff in combination with carrier sensing.

Most works on WSNs either consider an ad hoc network topology, such as S-MAC, or focus on the data delivery of the sensor nodes towards the sink. The work in [30] focuses on the delivery of data from the sink towards the sensors in the context of an infrastructure network topology. Such data transfers are often limited to configuration information or requests, where the inter-arrivals of the packets are considered to be larger than the packet transmission time. The root of this research topic can be found in the fact that the Access Points (APs) are usually not considered to be power constrained. Therefore, upstream data transfer is not deemed an issue since the AP is allowed to continuously keep listening for data packets. The downstream communication on the other hand requires the minimization of idle listening and overhearing to preserve the energy. As such, nodes should define a duty cycle, where the nodes are operational in one part of the cycle and are in a sleep state in the second part of the cycle. The proposed protocol, WiseMAC [30], is based on preamble listening. The concept is to listen for activity on the wireless medium at periodic intervals. If such activity is detected, and the packet is destined for the receiver, it maintains in its receive state, otherwise returns to sleep. An issue with this method is the length of the preamble. On the one hand, it limits the actual throughput of the network, while on the other hand, it presents an overhead at the receiving node. WiseMAC defines an alternative method of determining the length of the preamble in a dynamic manner, taking into account the network load. The AP is to learn about the sampling schedule of the nodes by extracting the relative time to the next scheduled sampling time from the packet acknowledgements. Since the clock drift needs to be taken into account in any possible manner, the preamble length needs to cover all those time differences. Therefore, the preamble length is a function of the inter-arrival time of the packets and the clock drift. If the function results in a preamble duration longer than the sampling period, the sampling period is used for the preamble duration. As a result, in high traffic conditions, the preamble is very short, while in low traffic loads the preamble is rather large. In order to minimize overhearing in such low traffic conditions, the preamble includes sections of data, containing the data header. Nodes search for the Start of Frame Delimiter (SFD) to detect an incoming packet, after which the destination address is verified. If the receiver is not the destination for the packet, it goes back to its sleep state.

3.2 Scheduled data transmissions

Time Division Multiple Access (TDMA) is a popular method in WSNs to alleviate the energy waste due to idle listening. Thanks to the precisely controlled operation, nodes are allowed to sleep during the time where no communication is required. A downside of the scheduled access is the increased complexity and the additional control overhead. Therefore, a large number of works is devoted to finding an efficient slot assignment algorithm. The discussed works are a selection of different slot assignment methods.

Note that some protocols, such as [31][32] and [33], either propose a flexible frame length, by either the dynamic adjustment of slot sizes or the number of slots, or announce the slot schedule of the following frame in a beacon. Often both methods are combined. Most of those works are not discussed in this chapter, since they present practical issues. The passing of information regarding the adjustment of the slot size and in most cases also the frame size to all sensors in the network needs to be carefully planned. The wireless medium is considered to be a lossy medium, which complicates the size adjustments, that is, nodes are not guaranteed to receive the new schedule. For the same reason is having an adaptive frame length a concept which poses practical issues. Some of the discussed works do make adjustments to either slot or frame size, however provide either a revert operation or ensure a secondary method that distributes the schedule information further, often in combination with CSMA channel access mode to prevent collisions during the slot access.

3.2.1 Distributed slot assignment

The slot assignment methods can be categorized according to the location of their assignment decisions. A first category makes use of a distributed slot assignment, where the decision of which slot to assign is made by each node individually. The process usually involves a certain information exchange with the neighboring nodes, some within a request and grant negotiation, others by means of control messages. It should be noted that the slot assignment is independent of the employed topology. Nodes ordered in a tree can also make decisions in a distributed manner, as one of the discussed protocols will show. The provided overview depicts only a small subset of the state of the art. The selection is based on the feasibility to implement the protocol on a sensor device without too many changes, some of the discussed protocols show similarities to the protocol that is discussed in Chapter 5, others present interesting features which could be used to enhance the protocol in Chapter 5, and others provide some alternative methods to achieve the same goal.

The problem of scheduling broadcasting slots is targeted in [34], which comprises the search for an interference-free schedule that provides an optimal throughput. The work identifies the NP completeness of the problem and proposes a heuristic for achieving a 'maximal' schedule. It has been assumed that all nodes have a uniform transmission requirement, that is, every node has an equal amount of data to transmit. The proposed method employs a slotted time, where the duration of a single slot matches the transmission time of a single packet, which are of constant length. Moreover, the identity of two-hop neighbors is assumed to be known by means of some higher layer protocol. In order to agree on a schedule that eliminates all possible collisions between transmissions, the two-hop neighborhood schedule information is required. Obviously, such requirement forms an issue, since the schedule needs to be derived from a schedule that requires other schedule information. As a means to resolve the issue, the protocol assumes a skeleton schedule, where each node within the two-hop neighborhood is assigned a single slot according to its identity. Part of the schedule of its two-hop neighbors is also known to the node, since it is aware that none of the two-hop neighbors are allowed to send at the same time. By exchanging the schedule with the other nodes, a more complete schedule is achieved. In order to determine the assignment of free slots, that is, slots that are assigned to a non-interfering node, a certain priority is assigned to the nodes.

The work in [35] considers a WSN as a network in which a large number of sensors are deployed in an ad hoc manner. Therefore the self-organization of such network is imperative. Since the recharging of sensors may be too costly, energy efficiency is also paramount. The proposed protocol, TRAMA, attempts to reduce the energy consumption, provide sufficient throughput and fairness while ensuring a collision free schedule. The protocol is fundamentally based on the exchange of information. A neighborhood discovery protocol gathers information about the two-hop neighborhood during a random access period. Since CSMA is employed during that period, the duration of the random access is made sufficiently long to provide a high statistical probability of receiving or transmitting their messages successfully. The remainder of the periodic cycle is made available for scheduled data transmissions. The last slot assigned to a node is used to notify its direct neighbors about its planned transmissions, indicating the expected receivers by constructing a bitmap which represents all neighbors. Nodes that are neither a transmitter, nor a receiver for a certain slot, can enter the sleep state in order to preserve energy. The slot assignment to nodes is determined by means of a hash function which determines the priorities of nodes based on the unique node id and the slot index. When a node has no data to transmit, it can forfeit its ownership of the slot, thereby allowing other nodes to take ownership of the slot. Since the protocol makes use of TDMA mode, synchronization of the cycles is required, which is not explicitly specified in the work. Since the transmission rate of sensor networks is rather limited, the clock drift is considered to be insignificant to the duration of a packet and therefore a simple synchronization protocol is said to suffice. It is assumed that some synchronization protocol operates during the random access period in order to ensure a sufficiently accurate timing. The work in [36] is based on this work, however, instead of basing the adaptiveness on the queue information of nodes, it is based on a more general application flow information, which is distributed during the random access period.

The work in [37] regards a sensor network as an intelligent entity, instead of just a data gathering network. The proposed protocol, AI-LMAC, considers the behavioral constraints of the application in order to adapt the network operation. The frame format, slot allocation and transmission method of the protocol are based on LMAC, which is a TDMA based protocol where nodes determine their slot assignment by means of two-hop neighborhood information. In LMAC, Each time slot consists of a Control Message (CM) and a Data Message (DM) section. The Control Message is transmitted by every node for the first owned slot, since nodes may own several slots. Neighboring nodes are expected to listen for the CMs and may enter sleep mode if the transmission is not directed to them. Beside controlling the early sleep of nodes, the CM supports the local time synchronization, provides an indication of the distance in hops to the sink, announces the slots that are considered to be owned by the node and its neighboring nodes, and provides an acknowledgement mechanism for successfully received data messages. Since the CM is responsible for the frame synchronization, the Control Messages are precisely timed. Thanks to the exchange of the occupied slots of its neighborhood, nodes can select a free slot in a distributed manner without causing conflicts. The AI-LMAC protocol includes small improvements over LMAC, however, the major enhancement is the introduction of a data management framework, which resides on each node. The considered application is environmental monitoring, where a set of heterogeneous nodes is used, that is, nodes can have different sensors attached to them. Nodes can appear and disappear from the network at any time. The sensor network is considered to be a large storage of information, shaped according to a tree topology, which can be queried by several environmentalists, even at the same time. Depending on the query, certain parts of the network might experience an increased data throughput compared to others. Therefore, the network should adapt itself based on the provided queries. In order to support the adaptivity, a Data Distribution Table (DDT) is defined for each sensor. The table is updated upon each packet arrival from one of its children, storing statistics of the traffic flowing through the network, such as type of sensor data, the originating region, etc. Based on this information, a more intelligent dissemination of the queries is made possible, instead of just flooding the network with queries. Based on the information gained from the tables, AI-LMAC is able to vary the number of slots a particular node owns depending on the expected data flow. Note that a node is able to deduce the relative needs of each of its children, however, it is not able to deduce its own relative need compared to its siblings. Therefore, the parent node is responsible for advising its children about the optimal number of slots. The child needs to consider the advise and make an attempt to follow it as closely as possible, depending on the number of free slots. Note that the number of slots of the child should not exceed the number of slots of the parent to prevent buffer overflows. The advice should start at the root node when a query is proposed to the network.

The work in [38] proposes a lightweight and localized algorithm to determine an available time slot in a dynamic manner. The algorithm operates in a completely distributed manner and is proposed to be used in combination with LMAC, where the schedule is assumed to be fixed. It is well known that the optimal slot assignment is an NP-hard coloring problem. The protocol therefore does not make an attempt to assign the minimum number of time slots, however targets to find a reasonable number that is sufficiently flexible to support the adding of new nodes and changes in the network density due to mobility. It is assumed that a frame has a fixed number of slots, which is a global network parameter. During the initialization state of a node, it samples the wireless medium in order to detect other nodes to which it can synchronize. When the synchronization is complete, the node waits for a random time before gathering network information to determine the available slots in its two-hop neighborhood. Every slot consists of a Control Message (CM) and a Data Message (DM). This Control Message takes a crucial role in the selection of a free slot. The transmitted bit vector, which is embedded in the CM, considers the controlled slots in the neighborhood of the node, allowing nodes to construct an overview of the slot usage in their two-hop neighborhood. Such overview provides sufficient information for the node to select a free slot.

The Advanced Medium Access Control (A-MAC) protocol is a TDMA based protocol designed specifically for low rate and reliable data transfer, while keeping in mind the requirements regarding power preservation [39]. The protocol is claimed to be a modification of LMAC. As such, each medium access time slot is composed of two messages: a beacon and data. Every node transmits a beacon at the beginning of its own assigned slot. Such beacon contains not only synchronization information, but also neighborhood and future data transmission information. As such, nodes are aware of which nodes will participate in the communication during the next frame. Nodes that are not engaged in the communication can enter a sleep state, thereby conserving energy. The TDMA slot assignment is decided in a distributed manner, based on the information distributed through the beacons. The slot assignment is a procedure comprising three phases. First, a node trying to join the network listens for transmitted beacons. A node that synchronizes to one of the beacon messages, remains in the synchronization state during a certain time in order to select the most fit beacon transmission to synchronize to. Second, after the synchronization phase, the node waits for a random number of waiting frames such that not all nodes enter the discovery stage at the same time. During this stage, the node gathers neighborhood information during a specified time interval and creates a bitmap of the assigned slots. From this bitmap, a random free slot is selected to be assigned to itself. Last, once a slot has been selected, it transmits its own beacon at the start of its slot.

The slot allocation mechanism, called DRAND, proposed in [40] and [41] is based on a channel allocation scheme, called RAND, used in mobile ad hoc networks. RAND arranges all nodes in a graph in a random total order and assigns to each of them the minimal color that has not been selected previously by one of its neighboring nodes. Due to the typical network size of WSNs, RAND is not particularly suited to be used in such networks. DRAND provides a randomized slot selection algorithm which complies to the requirements that a protocol for WSNs need to comply to. It is assumed that the slot size is selected such that they are sufficiently large to transmit a single packet. The number of slots is set to a fixed number per frame and the network is assumed to be synchronized. Moreover, all neighbors in the two-hop neighborhood are assumed to be known. DRAND operates in a number of rounds, during which each of the contending nodes is allowed access to the channel with a probability inversely related to the number of contending nodes. A node that has won the channel access lottery, sends a request message to all its neighbors. Those should all respond with a grant message for the requesting node to be granted access to a slot. The nodes that received the request are required to retransmit their grant packets until they are notified by the requesting node. Such notification can indicate the resign of the current slot selection round when a grant was not received by all neighbors within the current round. In this case the node has not selected a time slot. Otherwise, such notification can indicate the decision of which slot has been selected. When all a grant has been received from all neighbors, the requesting node selects the lowest time slot index that is not yet assigned to one of its neighbors. In order to prevent collisions during this setup phase, all nodes make use of a random backoff time before accessing the wireless medium. When all contending nodes have decided on a slot, the maximum slot size is distributed periodically to the global network, where the maximum slot number is transmitted in the owned slot. Nodes that have a different view of the maximum number of slots, adopt the maximum number of slots and forward this number.

Thanks to the collision free schedule that a TDMA access mechanism is able to provide, the waste of energy can be reduced in Wireless Sensor Networks [42]. The work proposes a Receiver-Driven Medium Access Control (RMAC), where the slots are assigned to the receiver, instead of the transmitter such as in most works. The protocol is claimed to prevent the broadcasting of explicit control messages or traffic schedules. RMAC is composed of three phases, the neighborhood discovery phase, the slot allocation phase and a data phase. During the first phase, one-hop neighborhood information is exchanged by means of broadcasting '*Hello*' messages, allowing all nodes in the neighborhood to construct a consistent view of its neighborhood. By employing DRAND, each node is guaranteed to be assigned a single slot, unique within its two-hop neighborhood. As such, the schedule is ensured to be collision free. The nodes are assumed to act as receivers during their assigned time slot and are sleeping during the remainder of the slots. The receiver requests in its assigned slot for a specific node to act as its sender by means of a 'textitRequest' message. While the sender acknowledges this request, the other nodes are able to overhear the messages exchange, thereby removing the sender from their list of potential senders. All receiver nodes will eventually select one of the nodes in their neighborhood to act as their sender. By the end of the time slot allocation phase, every time slot will have a designated sender and receiver node. During the scheduled data phase, all nodes either act as sender or receiver in their assigned time slots, or sleep. This scheme eliminates the need of the sender to inform their receivers of a future transmission. As an optimization, the receiver can return to its sleep state if no data is being received during a certain time interval. In order to alleviate the rigidness of the fixed TDMA slot assignment, the concept of slot stealing is introduced. Besides the primary assigned sender, a receiver is capable of selecting a secondary sender, which should perform channel sensing before transmitting in the slot in which its role is to act as secondary sender. If the primary sender is not using the wireless medium, the secondary sender is provided the opportunity to send its data. While the slot stealing does provide some flexibility to handle variable transmission loads, the performance is dependent on the pairing of the primary and secondary sender nodes. Therefore, RMAC proposes an additional slot reassignment scheme. Each sender is required to maintain the number of backlogged packets for n frames, while the receivers keep track of the number of times no data has been received. During the n^{th} frame, each sender broadcasts a control message in its assigned time slot, containing information regarding its backlog. Based on the received information of each node and the local information, each node decides on the reassignment of its time slot to another sender node. As such, lightly loaded sender nodes can be assigned a single time slot in multiple frames, while heavily loaded sender nodes can be assigned multiple time slots per frame.

As already mentioned in the introduction, there exist certain hybrid protocols, which employ both TDMA and CSMA channel access in order to make use of the interesting properties of both, without inheriting any of the negative characteristics. Z-MAC [43] is such a protocol, where the protocol is designed for its adaptability to changing traffic conditions; in low contention situations, the protocol behaves as CSMA, while it adopts more TDMA characteristics in high contention environments. The nodes in Z-MAC can operate in either Low Contention Level (LCL) or High Contention Level (HCL). The channel access time is controlled by means of time slots for both methods. The major distinction between the LCL and HCL mode is the slot access permission. During the default LCL mode, all nodes are allowed to contend for access to every slot. The slot owners are assigned a higher priority to access the slots thanks to a shorter backoff time. Non-owners can use the slot when the channel is considered to be free, that is, neither the owner, nor any other sensor makes use of the slot. The backoff, CCA and LPL mechanism of B-MAC are employed to allow for an efficient usage of the slots. In HCL mode, on the other hand, nodes are not allowed to use a slot which they do not own. The switch from LCL to HCL is controlled by transmissions of Explicit Contention Notification (ECN) messages to the destinations towards which high contention is experienced. It is noted that even though contention based channel access is used, the access times are controlled through the use of specific time slots. It is claimed that the protocol is able to work with a low-cost local synchronization algorithm, which makes use of the clock value of the neighbors, derived from received packets, to calculate a weighted moving average, which is robust against Byzantine errors [44]. The synchronization is made dependent on the traffic pattern of the nodes; the number of synchronization packets is a small percentage of the number of transmitted packets. Such synchronization is insufficient were it not that during the setup phase, a global synchronization, such as TPSN [45], is performed. Besides the global synchronization of the network, the setup phase also ensures a collision free slot assignment. By means of periodical ping broadcast transmissions, all nodes are able to construct a two-hop neighbor list. Such list is used as input to the DRAND protocol, which is responsible for the slot assignment. As such, a broadcast schedule is created where no two nodes within a two-hop neighborhood are assigned the same slot. Moreover, DRAND is capable of coping with a small number of nodes that join the network at a later stage without requiring the modification of the already existing slot schedule. The distributed slot assignment could result in nodes not employing the same time frame, that is, the periodicity with which nodes access the time slots. Since distributing the maximum slot number to all nodes in the network is a costly undertaking, Z-MAC allows each node to decide on its own cycle based on the local parameters.

As stated in [46], the conditions in emergency situations pose specific requirements on WSNs. The proposed protocol, ER-MAC, enables an energy efficient and delay tolerant operation during normal monitoring mode. In emergency mode, the protocol adjusts itself and gives priority to high packet delivery ratio and low latency over the energy efficiency. The switch from monitor mode to emergency mode is triggered by the detection of an emergency event. In such case, the nodes that detected the event broadcast an emergency message, which triggers the switch from monitor mode to emergency mode in neighboring nodes. The message propagates towards the sink, converting the one-hop neighbors of the path towards the sink. The remainder of the nodes in the network maintain their normal operational mode. For both the monitoring mode as the emergency mode two data queues are defined, one for high priority traffic, the other for low priority data. The data stored inside those queues are ordered according to their slack time, that is, the time remaining until the expiration of the packet delivery deadline. The order is used to determine the packets that need to be dropped in case the queue is full and new packets are arriving. During the monitor mode, the queue from which data is sent is the highest priority queue, unless the queue is empty. The slots in which the nodes transmit is determined by the slot allocation phase. Specifically for the emergency mode, four sub-slots are defined at the start of each slot. The four sub-slots are sufficiently large to transmit a MAC header, while the remainder of the slot is sufficiently large to transmit a data packet. If the owner of a slot has high priority data to send, it activates the transmission mechanism immediately. Neighboring nodes with high priority packets perform carrier sensing in the first sub-slot, t_1t, and when no high priority data from the owner has been detected, it contends for the medium by sending a slot request in sub-slot t_1. The owner replies with a slot acknowledgement in the same sub-slot. The owner of a slot with low priority data can access the medium if it did not receive any slot requests in sub-slots t_0 and t_1. Neighboring nodes with low priority messages need to verify whether the owner has no packets to send, and the neighboring nodes have no high priority messages before requesting access to the slot by contending for the medium during sub-slot t_3. Interesting about the protocol is that a tree topology is used, even though the slot selection is done in distributed manner. The setup method of the network consists of three phases, starting with the tree construction. Initially, before the tree structure is created, the protocol makes use of CSMA/CA to convey messages over the wireless medium. The tree construction is similar to the one in [47], which is discussed in detail in Section 3.2.2. The sink initiates the tree construction by flooding the network with topology discovery messages, containing the hop count, new parent ID and old parent ID. The message is used both for allowing nodes to select their parent and for a parent to detect joining or relinquishing children. The second phase, the slot assignment, is initiated by the leaf nodes and follows the reverse direction, that is from the leaf nodes to the sink node. Based on the neighborhood information gathered by the tree construction phase, nodes select a unique slot in its two-hop neighborhood and announce this information to their parents. Non-leaf nodes wait until all their children have announced their slot before selecting a slot for its own data, its synchronization broadcast slot and a slot for transferring the data of its children to its own parent. The last phase starts when the sink has received the slot information of all its children and starts the synchronization by broadcasting a synchronization message in its broadcast slot. Such synchronization message contains the ID of the sender, the current slot index, the TDMA frame length, the clock and the hop count. The receiver of such message is able to synchronize to the sender and is able to detect possible candidates to act as its parent in the event of a new node joining the network. Although the number of nodes that actively participate in the network also determine the TDMA frame size, ER-MAC provides a means to distribute the frame size changes to all nodes in the network upon the joining of a new node. In order to take into account the lossy character of the wireless medium and the fact that all nodes are required to switch to the new TDMA frame length at the same time, a countdown counter is included in the synchronization messages, specifying the time at which the new frame length will be enforced.

3.2.2 Centralized slot assignment

The centralized slot assignment usually makes use of the sink to make the slot allocation decisions, which distributes the TDMA scheme to all nodes. The sink is provided an entire view of the network, allowing it to construct an efficient slot allocation. The methods discussed in this section are only a small selection of the state of the art and are widely varying, such as genetic algorithms, breadth first search, graph coloring, etc. The algorithms are selected based upon similar features or goals as the proposed MAC protocol in Chapter 5, but also in order to provide an overview of alternative methods. Thanks to the centralized gathering of topology information, a complete view of the network is available. Nonetheless, often heuristic approaches are used since the search for the minimal TDMA cycle is proven to be NP-complete. It is also noted that in order to provide extra robustness, flexibility or features, often a contention based access method or carrier sensing is provided besides the existing TDMA scheme.

Although the work in [48] does not target sensor networks, battery operated devices are taken into account where the energy is considered as a critical resource. The proposed protocol, E^{2}MAC, considers the energy efficiency while maximizing the performance of the network. The performance is expressed in terms of the ability to provide support for certain QoS classes, which manipulate the available resources for multimedia traffic. The network architecture is infrastructure based, that is, the wireless stations of a single radio cell are supported by their base station, which is considered to be virtually resource unconstrained. Both the uplink and downlink traffic is Time Division Multiplexed (TDM), where the frames are of constant length, consisting of multiple slots. The mobile nodes and the base station are synchronized such that the frame start of the nodes is aligned. The operation of the protocol requires three different slot types: traffic control, shared, and data slots. The traffic control slots are used by the base station to direct the traffic of all nodes by broadcasting the schedule. Since the network operation and QoS depends on the information embedded within those packets, they are protected with an error correction mechanism. The data slot is used to transmit data and is ensured a collision-free operation. The remainder of the slots are considered shared slots, from which the first half of the slot can be used by the base station to transmit downlink calls, and the second half can be used by the nodes to make connection requests. The connection requests are successful if no contention occurred and the message successfully arrived at the destination. While establishing a new connection, the requested service class and QoS parameters are announced to the base station, which is responsible to translating these requirements into network parameters such as number of data slots, frequency and delay. Depending on the current traffic load in the radio cell, the base station determines whether it is capable of accepting the new connection. If sufficient resources are available, the node is notified of the connection-ID assigned to the new connection. Updates of reservations can be requested by piggy-backing information on the data packet. The scheduling of the data slots is arranged such that the energy consumption is minimized, by reducing the number of required transitions.

The work in [49], the PRNET model, focuses in finding the smallest possible TDMA frame, while ensuring a conflict-free concurrent transmission of multiple nodes during a single slot. An attempt is also made to maximize the channel utilization with regard to the throughput. The nodes are assumed to be frame synchronized by means of a single clock source and a single slot size is considered sufficient for transmitting a packet. The idea is to find the minimal TDMA cycle where all nodes are provided with an opportunity to transmit data. Since the targeted problem is proven to be NP-complete, a Genetic Algorithm (GA) is used to find a near optimal solution. Common concepts used in GA are chromosomes, selection, crossover, and mutation. Two algorithms have been verified, the standard GA and a modified GA. A GA consists of three steps, the encoding and initial population, the fitness evaluation and selection, and the crossover and mutation step. The differentiation between the two methods is made in the final step. Since a TDMA cycle can be expressed as a binary code, the encoding step is straightforward. The chromosome is initialized by a TDMA cycle with N time slots, each of which is allocated to a different node. By random permutations of 1 to N, several such solutions can be generated by providing a different slot assignment to the nodes. The quality of the different competing solutions is measured by the length of the TDMA cycle, provided the solution is valid. The crossover operation, which involves swapping part of the TDMA sequence with another chromosome, could generate offspring that would violate certain constraints, thereby resulting in invalid solutions. Such solutions are penalized by means of a penalty function. All generated solutions are ranked according to the fitness property. A tournament selection is used for selecting the chromosomes to be used for generating the next generation. This process is repeated P number of times to select P members of the next generation. This method works for a low number of nodes in the network, however, for larger networks, the approach fails. The modified GA approach introduces some knowledge about the constraints in the crossover operation, such that no invalid combinations are generated. This approach led to a more successful method, where also larger networks could be modeled.

The work in [50][51] considers Wireless Sensor Networks where all sensors periodically generate data. The network is constructed according to a tree topology, where all sensors are assumed to be energy constrained devices, whereas the sink is not subjected to such constraints. As a consequence, the packets originating from one of the nodes within the network need to be transfered in multiple hops to the sink, while the sink is sufficiently powerful to cover the whole network with a single transmission broadcast. The proposed MAC protocol, PEDAMACS, which is a TDMA based protocol, defines a topology learning phase, a topology gathering phase and a scheduled data transfer stage. The '*Topology Learning Phase*' is initiated by the sink, which transmits a topology learning control packet containing its current time and the next scheduled control packet. As such, all nodes are able to synchronize to the sink, while at the same time they are being informed of any future control packet transmission. Following this control packet, the sink floods the network with a tree construction packet, containing amongst others the number of hops the packet has traversed. Each node increments the number of hops and forwards the packet. As such, nodes obtain a list of potential parent nodes, of which the node with the lowest hop count is selected to act as its parent. Thanks to the flooding of the network, each node is able to construct a table of all its neighbors and interferers. The '*Topology Collection Phase*' is initiated in a similar manner as the former phase, that is, by the broadcast of a control packet by the sink. The control packet, a topology collection packet, also includes the current time and the next time a control packet is to be expected. The reception of such packet triggers the transmission of a '*Local Topology Packet*', consisting of the parent, neighbors and interferers of the sender. The packet is forwarded from child to parent until it reaches the sink. Both phases employ a random backoff wait time in combination with carrier sensing since no schedules have been defined yet. In order to ensure the successful reception of the '*Local Topology Packet*', an implicit acknowledgement mechanism is employed, where the sender tries to overhear the transmission of the packet by its parent. The collected topology information enables the sink to create a schedule, where each node is assigned a single slot in order to promote fairness. The schedule is constructed such that the packet latency, that is, the time between the generation of the packet and its arrival at the sink, is minimized. Since this optimization problem is known to be NP-complete, this work proposes a polynomial-time scheduling algorithm that achieves a delay proportional to the number of nodes. Based on the collected topology information a connectivity, interference and conflict graph can be constructed. The conflict graph includes edges between node pairs that should not transmit at the same time, such as when two children of the same parent should not transmit in the same time slot in order to prevent collisions. Based on those graphs, a linear connectivity and interference graph is constructed, where a single node represents all nodes at a certain level in the original connectivity graph. The interference graph includes edges where an interference is present between a node at a certain level and any node of another level in the original interference graph. The linear network is then colored such that two nodes with the same color are able to send at the same time without causing collisions. The resulting schedule is broadcast by the sink at the start of the scheduling phase.

Based on the conjecture that TDMA protocols are more interesting for WSNs, the work in [52] proposes two methods for multi-hop TDMA scheduling. The schedule needs to be designed such that not only collisions are avoided, but also the number of slots is minimized to optimize the overall latency. Multi-hop TDMA scheduling differs from the single hop scheduling in that spatial reuse may be possible, that is, multiple nodes may be able to transmit during the same time slot without interfering each other. This work first proves the NP-completeness of the depicted problem and then proposes two centralized heuristic approaches based on conflict coloring to generate a minimal schedule. The considered network comprises a single Access Point and several sensor nodes that periodically generate data. The network is represented by a graph, which forms a tree structure. Since all data is destined towards the AP, all packets are sent to the immediate parents in the tree. The interference graph, indicating the nodes of which the transmissions would interfere with each other, is assumed to be known. The first method is based on a classical method employed in ad hoc networks, modified specifically for Wireless Sensor Networks. The algorithm constitutes of two parts, the coloring of the conflict graph and the scheduling of the links based on this coloring. As regards the coloring, any protocol, which ensures that two nodes are assigned a different color if their combination is marked in the conflict graph, can be employed. The work selected the heuristic vertex coloring as preferred protocol. Vertices are colored sequentially with a selected color, taking into account the already chosen colors in the neighborhood of the vertex. Usually the nodes are first ordered according to a decreasing order of degree, since the high order degree vertices pose a higher number of color constraints. The algorithm then assigns the smallest color to the nodes such that there exists no edge in the conflict graph between nodes with the same color. As a result of the spatial reuse of slots, the number of slots within a frame is limited to the number of employed colors. A mapping is therefore made between the colors and the slots within the frame. The second method involves coloring based on the node levels within the tree topology and is comprised of three elements. First a linear network is constructed corresponding to the original network. A linear conflict graph is constructed where the edges indicate whether the transmission of a node at a certain level interferes with a node of a different level in the original network. Any coloring algorithm can be employed for the constructed network conflict graph. The slots are allocated to the nodes of different levels such that each level of the tree is capable of forwarding at least one packet to the next level during a single frame. Like in the first approach, the number of slots is limited to the number of colors, since nodes at different levels with the same color are expected not to interfere with each other. The results indicate the advantage of using the level-based scheduling in a network where a larger number of packets are being transmitted at the higher levels of the topology. However, in networks where an equal transmission flow is present in the network, the direct scheduling of the nodes shows an improved performance. Since both approaches require information about the full network topology, possible approaches towards a more scalable solution have been investigated. A possible improvement could comprise the organization of the network in clusters.

Wireless Sensor Networks exhibit usually a typical communication pattern, that is, a many-to-one traffic pattern to one or multiple sinks. The work in [53] focuses on the funneling effect of such pattern, where nodes closer to the sink experience an increased number of packet collisions and congestions compared to nodes farther from the sink, should a contention based access method have been used. The nodes closest to the sink, each of them within a small number of hops from the sink, present a larger loss of packets and a thereby associated higher energy consumption due to retransmissions. Therefore, the work presents F-MAC, in which the nodes closest to the sink operate in a scheduled mode, that is, TDMA, whereas the nodes farther away operate in CSMA mode. The TDMA mode operation is triggered by a beacon sent by the sink. Such beacon contains the superframe duration, TDMA duration and beacon interval. A superframe is composed of a CSMA period and a TDMA period, of which the TDMA duration can be adjusted dynamically upon demand. All nodes operate by default in CSMA mode, unless such beacon has been received. The sink regulates the power by which the beacon is sent and thereby regulates the number of nodes that operate in TDMA mode. The region in which nodes access the medium according to the TDMA schedule is called the intensity region and the nodes residing in this region are called f-nodes. Besides the triggering of the TDMA medium access operation, the beacons are also employed to ensure a synchronization of the receivers, similar to Reference Broadcast Synchronization (RBS) [54], which is discussed in more detail in the following chapter. In order to measure the required dynamical adjustment of the size of the intensity region, each packet is marked by the first f-node it passes, which is designated as the path head. Each subsequent node increases the number of hops counter of the packet. The sink maintains a table of possible paths and their associated traffic rate. During each superframe, the sink calculates the weighted moving average of the measured traffic rate per path and uses this information to allocate time slots per path. In order to optimize the slot allocation, F-MAC makes use of a simple rule that enables spatial reuse, that is, nodes that are separated by more than two hops are not assumed to interfere with each other. The schedule is not transmitted in the beacon packet in order to minimize the overhead. When a schedule is to be conveyed, a schedule bit is marked in the beacon packet, notifying all nodes in the intensity region about the impending schedule transmission. The schedule packet is sent with the same transmission power as the beacon message such that the entire intensity region should be capable of receiving the schedule. The schedule itself contains the path identification, by means of the path head, and the number of slots assigned to the path. Since all nodes maintain a list of paths they belong to, each node is capable of determining its assigned time slot. Since the wireless medium is considered lossy, F-MAC incorporates mini-schedules, which are regularly transmitted by the f-nodes to announce the used schedule to inform other nodes. Some optimizations are taken into account to improve either the robustness or power efficiency. To safe-guard the robustness of the protocol carrier sensing is performed at each packet transmission, even though the TDMA slots are scheduled. In order to improve the power efficiency of the carrier sensing, Low Power Listening (LPL) is employed, such as in B-MAC. The nodes operating in contention based access mode are required to use long preambles. However, the nodes operating in TDMA access mode can afford smaller preambles, thanks to their scheduled spectrum access.

Despite the specific requirements and resource constraints of WSNs, some protocols, such as HyMAC [55], are designed to provide a high throughput and bounded latency across multiple hops. Thanks to a TDMA schedule, the protocol is able to reduce the waste of energy to a minimum thanks to the predictable sleep interval and implicit collision prevention. The required synchronization of nodes is assumed to be attended by one of the established synchronization protocols. The TDMA cycle consists of a fixed number of slots, where each slot is designed such that it provides sufficient time for the nodes to transmit the maximum packet size. A predefined number of slots are dedicated for scheduled transmissions, while the remainder of the slots can be shared between the nodes by employing a contention based access mode. In order to limit the overhead of the contention based access mode, Low Power Listening (LPL) is used. The contention slots are used both by nodes joining the network and for publishing the neighbor list that nodes have constructed. The sink is able to collect all transmitted neighbor lists thanks to the forwarding of such information towards the sink by each node. By means of the derived global view of the network topology, the sink is able to compose a TDMA schedule for the entire network. The construction method is a heuristic algorithm, since the construction of a minimum delay schedule can be reduced to the NP-complete distance-two graph coloring problem. The derived tree is traversed in a Breadth First Search (BFS) manner, where each node of a single level is assigned a default slot, which increases with the level. Possible interferences between one-hop and two-hop neighbors on the same level are verified. If two siblings are assigned the same slot, one of the siblings is assigned a different slot. If the interfering nodes are no siblings, one of the nodes is assigned a different frequency, such that the nodes are still able to send the data concurrently, only at different frequencies. When all nodes are assigned both a slot and frequency, the slot assignment is reversed, such that the slot number of children is lower than the slot assigned to their parent. Each node in the network is notified of the schedule by the sink.

Since WSNs can contain a considerable number of nodes, often a protocol is required which attempts to minimize at the same time the data gathering time and the energy consumption. The work in [47] considers such protocol where the goal is to minimize the time required to perform the data gathering in a fair manner of all sensors in a single round. By employing a scheduled medium access method, the energy efficiency is maximized since the sensors are placed in a sleep state when they are not assumed to be active. The specific use case for which the protocol is developed makes certain assumptions, such as the network topology being a tree, all nodes being synchronized, all nodes report periodically to the sink, etc. It is noted that the nodes forward the measured data in their raw form, that is, not aggregated, which is claimed to be justified in environments where the correlation between the different types of data measurements is low and unpredictable. The work comprises both a theoretical analysis on the most optimal scheduling method, such that the time required to reach the sink is minimized, and a practical protocol design. In the theoretical analysis is both the homogeneous, that is, every node is assumed to have an equal amount of data to send, and heterogeneous case considered. The analysis focuses first on the subtree data gathering, where different topologies are considered, such as a linear 'tree', a binary tree and a more complex tree structure. By the information gained from the more trivial tree constructions, an algorithm is designed to schedule the data within the more complex tree, which can be observed as a collection of more basic subtrees. Since a node is constrained to be either transmitting or receiving, the transmission and reception functionality of nodes is alternated within each level of the respective subtrees, resulting in a wavelike pattern. Since the sink of the tree is not constrained by the simultaneous transmission and reception constraint, the algorithm to gather the data of the whole tree is constructed such that data is gathered alternately from the largest subtree and one of the other subtrees, which are scheduled according to their size in a decreasing order, until the largest subtree has been depleted. It is claimed that the order in which the subtrees are sorted does not matter, except that in the descending order a reduction of the maximum buffer size may be achieved. The data collection from any remaining subtrees is scheduled continuously, one after the other, until all data is collected. In the case of a heterogeneous data transfer, the required number of data slots is represented by means of a fractional number based on the number of transmitted bits and the slot size. The size of the subtree is redefined as the number of required data units, while the algorithm remains the same. The second part of the work consists of incorporating the obtained algorithm for the minimum reporting interval in a protocol design. The protocol consists of two phases, a tree construction algorithm and a slot assignment algorithm. During the tree construction phase, each node selects its parent base on the relative distance between the nodes, such that the neighbor with the minimum depth is considered as its parent. Such method is considered to guarantee a minimum-hop path from each sensor to the sink. Moreover, such minimum-depth can reduce the interference between neighboring communications. The tree construction is initiated by the sink, which sends a message containing its level, source ID and parent ID. The nodes receiving this message either accept or reject the sender as their parent, based on its level in the tree and signal strength of the received message. Upon selecting a parent, the node broadcasts its own message containing its level, source ID and parent ID. The parents are notified of the parent selection by the overhearing of the broadcasted messages of their children. In order to notify the parent that a node has switched parents, the message includes a previous parent ID field, indicating that the parent should remove the child from its children list if the field matches its own ID. When a node does not receive a message from any potential children within a certain time, the node assumes it is a leaf node and initiates a tree size notification, that is, it sends a message to its parent indicating its tree size. The parent updates its own tree size each time it receives such update from one of its children. As soon as all children have sent their respective tree size, the node sends its own tree size notification to its own parent, until the complete tree size is propagated to the sink node. The slot assignment algorithm alternately schedules the data transmissions on the longest path in the first and second slot. The remaining paths are scheduled in a reverse manner and the start of the slots is shifted in time in order to allow transmissions from the subtree. The decision on whether the node is considered as being a member of the largest subtree is based on a message transmitted by each parent, providing a list of its children ordered according to their subtree size. As such, each node is able to determine its own duty cycle based on its tree size.

3.2.3 Cluster based slot assignment

Cluster based slot assignment is a means to avoid as much interference as possible and maximize the slot reuse. By segregating the entire network in smaller network entities, called clusters, often a more simple slot assignment algorithm can be used, without requiring knowledge of the entire network. However, the slot allocation for all cluster members is typically determined by each respective cluster head. This section merely mentions the subject of clusters, since they might provide an added value towards the protocol described in Chapter 5, but since the protocol does not make use of them, the subject is limited to the description of only a few clustering protocols.

The organization of the network in clusters is claimed to provide an improved energy efficiency [56], which is demonstrated in the discussed protocol, Low-Energy Adaptive Clustering Hierarchy (LEACH). It is noted that the protocol assumes a homogeneous network, that is, all sensor consume their energy equally fast and have an equal amount of energy to their disposition. In order to distribute the energy load evenly among the available sensors, the protocol ensures a local decision system for determining the cluster head, while providing a randomized rotation of the cluster head functionality. Since the cluster head needs to transmit the assembled data over a long distance to the sink, the data is aggregated and compressed to reduce the energy consumption of such transmission. Since the cluster head functionality needs to be rotated regularly, LEACH operates in rounds. Each round starts with a setup phase, where the cluster heads are determined in a distributed manner. For each type of network, there is an optimal number of cluster heads, such that not too many nodes need to transmit directly to the sink, while still providing a sufficient coverage of the network. This optimal number of nodes is called the desired percentage of cluster heads, P. Based on this number and the current round, each node decides internally whether or not to become the next cluster head. A node that has been a cluster head during the last $1/P$ rounds is not eligible to become a cluster head again. As the number of rounds increases, the probability to become a cluster increases for each node that did not yet assume this functionality during the last $1/P$ rounds. When all nodes have assumed the role of cluster head, all nodes are again eligible to become cluster head. Each node that assumes the role of cluster head announces itself by broadcasting a cluster head advertisement message. The broadcasting is done by means of a CSMA type of channel access. The non-cluster heads are required to listen for the advertisements during the setup phase. Based on the received signal strength each node determines its cluster head and informs its cluster head of the new member of the cluster. After having obtained all cluster members, the cluster head decides on a TDMA schedule for all members and broadcasts it to its members. After the setup phase has been completed, the data dissemination phase starts, where each node follows the assigned TDMA schedule and transmits data by means of a minimal amount of energy, based on the received signal strength of the advertisement. Since interference between clusters can happen, the work proposes the additional use of CDMA codes. That is, each cluster head selects a spreading code randomly and informs its members of the selected code during the setup phase. All members are required to transmit using this code, while the cluster head filters the received energy by means of the selected spreading code. When all data from the sensors has been received by the cluster head, it performs signal processing functions to compress the data into a single signal, which is then transferred to the sink.

Instead of using a statistical method to select the cluster heads in the network, the work in [57] makes us of the residual energy of the nodes. During the setup phase the base station broadcasts a 'Hello' message across the entire network, thereby allowing each node to determine its approximate distance to the base station, based on the received signal strength. During the cluster head selection phase, nodes decide with a probability T to become candidates for cluster heads and broadcast a cluster head challenge message to all cluster head candidates within a certain range. Any cluster head candidate which receives such message from a node with more residual energy, forfeits the competition. As a result, only the candidates with the highest residual energy can become cluster heads. After the cluster head election, all cluster heads send advertisements across the entire network to allow plain nodes to select their cluster head. Based on the signal strength of the received advertisement and the distance of the cluster head towards the base station, nodes make a selection to which cluster head is the most appropriate for them. Since cluster heads located farther from the BS will consume more power to forward all data to the BS, such clusters should accommodate fewer cluster members.

Whereas the previous works consider a homogeneous network, where all nodes have an equal amount of energy, the work in [58] proposes the inclusion of a fixed number of high power cluster head nodes. The entire geographical area is divided into hexagonal cells. On the intersection of the vertices with two other cells a high power cluster head is placed. The regular sensor nodes are deployed in a random manner within this hexagonal area. The membership towards a certain cluster is determined in a similar manner as in the previously discussed protocol, that is, by means of the received signal strength. However, instead of directly communicating to the sink, the cluster heads communicate in a multi-hop manner with the sink. Since this communication might cause interference with the intra-cell communication, that is, the data forwarding of each sensor to the cluster head, FDMA is used. Each cluster head is assigned a different frequency, such that no interference is caused within its two-hop neighborhood. The cluster members share this frequency assignment. In order to prevent collisions between the cluster members, a TDMA schedule is proposed. The schedule consists of only four time slots that need to be shared between one fourth of all cluster members. The allocation of the sensors to one of the slots is based on the two least significant bits of their ID. The cluster head iterates over the list of members that share the specific slot and polls each of them one at a time whether one of them has data to send. As such, a considerable reduction in energy consumption can be achieved.

3.2.4 Position based slot assignment

Slot assignment based on the relative or absolute position of the nodes is usually a specific approach of distributed slot assignment. The nodes decide on the slots they are using based on local information, which is in this case its position towards either other nodes or the sink. For certain use cases, a disadvantage could be that all nodes should be aware of their position or should maintain a list of locations. This section provides two examples of MAC protocols where such methods are employed. A more in depth discussion regarding this type of protocols is out of the scope of this work.

The work in [59] proposes a TDMA MAC protocol, GMAC, where the schedule is determined locally, based on the geographic location of its neighboring nodes. The position information is assumed to be known or can be obtained by means of GPS or any other equivalent method. It is noted that GMAC does not require an accurate positioning method, it simply requires the positioning information to be consistent in the two-hop neighborhood of the nodes. The TDMA frame, called a super-cycle, has a specified duration t_{sc}. Each super-cycle is divided into c cycles of length $t_c = \frac{t_{sc}}{c}$, where a single cycle represents a fictive rotation of 360 °of a node. A node therefore makes c rotations during a super-cycle. Since all nodes are assumed to be synchronized, each node can start its rotation at the same time in the same direction with the same speed. In order to control the nodes that can be communicated with and the direction of the communication, a cycle is divided into s time slots, each of duration $t_s = \frac{t_c}{s}$. The fixed slot durations ensure a deterministic delay. By relating the fictive rotation of a node to the duration of a cycle, a link has been made between the relative position of a node and time. As time progresses, the angle, relative to the transmitter, increases. A node positioned on this angle can accept packets from the transmitter during that specific time slot. During a time slot only a single packet is transmitted, that is, the slot only allows communication in a single direction. The acknowledgement therefore needs to be sent in the slot where the receiver's angle is aligned to the transmitter of the packet. The synchronization is maintained by each packet transmission. Each packet is timestamped in the MAC, eliminating any possible latency from higher layers. Likewise, the arrival time of the packet is stored in the MAC, such that variable delays of the upper layers do not have any effect. When the packet is earlier than expected, the receiver's clock is updated to the sender. In the reverse case, the sender is updated by including the time difference in the acknowledgement, allowing the sender to update its clock according to that of the receiver. In order to ensure a consistent location information, a specific periodic super-cycle is defined where all nodes broadcast a beacon containing its own position. Any free slot can be used within the super-cycle to broadcast the information. As a result of all nodes broadcasting their position information, nodes are able to create a list representing the position information of nodes in their neighborhood. The following super-cycle is used for the broadcasting of such list, ensuring a consistent two-hop neighborhood information. In dense network conditions, multiple nodes might be assigned to a single slot. To remedy the situation, the deployment area is divided into cells of equal size. From each individual cell a node is allowed a communication opportunity during a unique cycle within the super-cycle, that is, each cell during its own cycle. Since nodes might have a different data rate, GMAC includes the possibility of slot stealing. Access to the slots is arranged by means of priorities. The priority is determined based on the distance between the most recently assigned slot to node n and the specific slot it is trying to steal. As a result, the owner of the slot has the highest priority. A neighboring slot can be stolen if the node does not detect any transmissions during a specified SIFS time.

The work in [60] proposes a MAC protocol, DGRAM, which takes into account the energy efficiency of the WSN and ensures a deterministic packet delay. The protocol makes use of the location information of the nodes, ensuring a TDMA based access based on the radial distance to the sink and the angular distance of the nodes from the sink with respect to the geographical North axis passing through the sink. It is claimed that most TDMA MAC protocols consider a centralized slot allocation, resulting in a scheme which is difficult to scale and requires a considerable amount of control overhead. DGRAM on the other hand, ensures a distributed slot allocation method, reducing the number of control messages. The reduction in energy consumption is established by the deterministic sleep pattern of the nodes, which is enforced by the TDMA scheme. In order to enforce the timeliness of the time slots, the protocol does not commit extra resources to time synchronization, since it is assumed that all nodes are synchronized by means of an out-of-band synchronization method. A specific requirement is placed on the position of the nodes, which are assumed to be deployed with uniform density around the sink, which is positioned at the center of the network. Each node is assumed to have obtained its own relative position to the sink by means of any of the established location information gathering methods. Since the nodes closer to the sink need to be able to cope with the data transmission of multiple nodes, data aggregation is assumed, where the data from the node itself and its lower positioned nodes can be aggregated in a single packet. The channel access time is broken down into super-frames, each of which consists of sub-frames. Likewise, the WSN is subdivided in tiers, based on the radial distance from the sink. Each tier is represented by a sub-frame. Since the reuse of slots is encouraged, thereby limiting the end-to-end delay, every N tiers, the same sub-frame is used. Each sub-frame is subdivided into a number of blocks, based on the angular position. Each block is assigned a sub-sub-frame, which in itself consists of a number of slots. Each node is assigned a single slot during the operation of DGRAM. Interference between neighboring blocks is deemed possible, therefore, an alternating slot allocation approach is used,that is, slot reuse is possible for nodes that are positioned two blocks apart. Based on the position of the node, each node is capable of determining its own tier and block. Nodes within a block make use of an ordered list containing the location information of the other nodes belonging to the same block to determine their respective slot. The exchange of this location information is done once during the setup of the network. A node broadcasts its location information in combination with a Time-To-Live (TTL) field. considering the TTL field has a non-negative value, the receiver of such packet forwards the information after having decremented the TTL field. The

TTL value is selected such that every node within the block is able to receive the location information. Any node not belonging to the same block is considered to discard the received information. Note that the size of a tier depends on the node density. Therefore, the super-frame length depends on the size of a tier. Since some tiers can send concurrently, that is, the same sub-frame is used to service nodes from different tiers, the maximum tier size needs to be taken into account to determine the size of the super-frame. Based on the distributed algorithm to determine the tiers and blocks, any node is able to determine the sub-frame and slot in which it is allowed to transmit. Since any node is able to deduce the transmit time, the receiving nodes of the higher tiers are able to determine the possible receive slots, where it might receive a packet from one of the nodes that are positioned in its neighborhood on a lower tier. Thanks to the hierarchical transmission and reception, no routing protocol is required, the packet is automatically forwarded towards the sink.

3.3 Latency efficiency through Wakeup patterns

Whereas the previous sections targeted protocols and their channel access method to avoid energy waste, this section describes protocols that consider the throughput and latency of the network. Interesting is that all discussed methods are designed as a solution to the long latencies introduced by duty cycling, such as in S-MAC. In order to counteract the negative effect on the latency, the transmissions and receptions of the packets are scheduled in a specific manner, such that a path is created through which a packet can be immediately forwarded over multiple hops.

The work in [61] proposes a MAC protocol, DMAC, which considers not only the energy efficiency of the network, but also the throughput and the latency of data transmissions. The work discloses several disadvantages to existing synchronous duty cycle approaches. A well known issue is the introduction of a long latency, also known as the sleep latency. Moreover, a fixed duty cycle provides no adaptation to traffic variations. The cycle needs to be selected to factor in the highest possible traffic flow, resulting in a significant energy waste during low traffic conditions. As a result of the synchronous character, the transmission of all nodes starts at the same, thereby increasing the probability of collisions. Protocols which adopt a dynamic duty cycle, such as T-MAC, exhibit the data forwarding interruption problem. That is, since overhearing is used to determine whether the medium is still actively used to determine the duration of the active period, the forwarding of data is interrupted. The nodes that are located several hops further are not aware of the channel conditions that usher a node to remain active. Their active period duration is determined based on local conditions. DMAC defines a staggered active/sleep schedule to eliminate all previously mentioned issues. The network is assumed to be synchronized by one of the established synchronization protocols (see Section 4.4). The sensor nodes are assumed to be more or less static and are organized according to a tree, where all data is directed towards the sink. The time allotted for transmitting and receiving a packet is sufficient for a single packet to be transfered over the medium. Each node is following the same cycle of receiving, transmitting and sleeping, only with a different offset compared to the frame start of the sink. As such, the transmit slot of the child node overlaps with the receive slot of the parent, allowing the traversal of a data packet directly towards the sink. Since nodes on the path wake up sequentially to forward the packet to the sink, the sleep latency is eliminated. In order to avoid collisions, the transmission of packets is preceded by a variable backoff time accompanied by carrier sensing. The general scheme is designed with the idea that a single packet is being transmitted. To cope with multiple packets that are available in the transmit queue, the duty cycle needs to be increased. The node therefore flags the more-data bit in the MAC header, indicating the request for an additional active period. Since multiple children might require to transmit data to the same parent, while only a single receive slot is available, DMAC proposes 'data prediction' to ensure a timely data transfer. The parent anticipates that some of its children did not have the opportunity to send data and therefore employs an extra receive slot. The slot has a fixed offset relative to the last transmission, allowing the children to deduce the transmission opportunity.

A different method to achieve the same goals, that is, maximizing the throughput and energy efficiency while minimizing the latency, is proposed in [62]. The Routing enhanced MAC protocol (RMAC) employs cross-layer routing information such that the transfer of a data packet over multiple hops is made possible in a single operational cycle. The operation of the protocol is organized in frames, which consist of a sync, data and sleep period. The sync period is employed to provide frame synchronization between the nodes. The work does not elaborate on the specific methodology, except that it may be one of the existing synchronization methods such as [45][54]. The data period is employed to request access to the medium, not unlike the purpose of the RTS/CTS message exchange in IEEE Std 802.11. RMAC employs PION packets instead of RTS and CTS messages and employs only a single packet for both the request and confirmation. A node needs to send a PION packet to its next hop towards the final destination. The next hop information is assumed to be provided by the routing layer. Upon reception of such PION packet, it makes a reservation for the reception of the data packet and construct a new PION packet containing amongst others the next hop address and the previous hop address. The transmission of such PION packet should be received both by the previous hop and next hop node. As such, a chain of reservations is made until either the destination address or the end of the data period is reached. The data transmission will start during the sleep period. Note that not all nodes in the transmission path need to be awake at the same time, since the packet might need to traverse a number of hops first. The nodes maintain their sleep state until the time has arrived to receive and possibly immediately transmit the packet. Afterwards the node enters its sleep state again. In order to prevent collisions, the reception of PION packets triggers the updating of the Network Allocation Vector (NAV). Since the PION requests for future transmission, not a single NAV duration is modified, instead three NAV segments are stored. The segments consider the actual PION transmission, the data transmission and the acknowledgement of the data.

The introduction of duty cycles is deemed to be a most efficient approach against idle listening. However, the latency induced by the duty cycle forms a cumulative latency in multi-hop networks. To counter the long latencies, RMAC includes a method to transfer a packet over multiple hops during a single cycle [62]. However, by establishing such method, RMAC also introduces an extra time segment of idle listening. During the data period, resource allocations are requested, such that transmission and receive slots are made available during the sleep period. Since all nodes need to listen during the data period, this incurs a small period of idle listening when no data is available. The MAC protocol proposed in [63], LO-MAC, proposes an improvement of RMAC by introducing a small period of time before the data period, that is, the carrier sensing period. During this period of time, nodes that have data to transmit emit a busy tone, whereas potential receivers try to detect whether the medium is idle. If the medium is considered idle, all nodes are allowed to enter sleep mode, thereby reducing the idle listening which was introduced by the data period. If the medium is considered busy, the nodes stay awake to complete the data phase as defined in RMAC. The eventual data transmission is also based on RMAC, where the data is transmitted during the scheduled slots. However, while RMAC defines an explicit ACK arrangement, LO-MAC employs an implicit ACK arrangement, where an energy detection is performed of the first symbol of the forwarded packet to determine whether the packet has been received successfully. The final node is required to transmit an explicit ACK in order to notify its sender.

Unlike the previous works, the work in [64] is not designed to improve the throughput. The protocol focuses on the end-to-end latency and the energy efficiency. The general belief is that sensor networks are used to detect rare events that require immediate attention. The popular approach of introducing a duty cycle for reducing the energy, thereby allowing the nodes to periodically enter a sleep state, results also in a higher latency. It is claimed that, although research has been performed for new wakeup methods, these methods are considered impractical. Therefore, new efficient scheduled wakeup schemes should be investigated that can be considered cost-effective, while at the same time guaranteeing the delay and energy efficiency. The work assumes the presence of a time synchronization protocol ensuring a consistent notion of time between the nodes. Each node in the network is represented by a node in a graph, where the edges identify the possibilities to communicate. An initial connectivity graph is constructed during the network initialization by the base station, which can be adjusted by occasional updates. It is assumed that such connectivity graph remains fixed. Each node is assigned a level, indicating its minimum number of hops to the sink, by means of a Breadth First Search of the connectivity graph. The work performs an analysis on the energy efficiency and latency between possible wakeup schemes, such as '*Fully Synchronized Pattern*', '*Shifted Even and Odd Pattern*', '*Ladder Pattern*', '*Two-Ladders Pattern*' and '*Crossed Ladders Pattern*'. The work proposes a novel class of wakeup methods, called multi-parent scheme, which can be used in combination with any of the previously mentioned wakeup schemes. The idea is to allow a node to have multiple parents, each of which belongs to a distinct group. Each group is scheduled to operate during a different frame, which is defined as a single wakeup period. The number of groups g

determines the number of frames that constitute a cycle. By increasing the number of parents, either the parents need to wake up less frequently, or the wakeup pattern is increased by g, resulting in a lower latency while preserving the consumed energy. An important part of the protocol is therefore the parent selection, which is defined as a graph coloring problem and can be assumed to be NP-complete. Therefore, a heuristic algorithm, which operates on the connectivity graph, is proposed to solve the parent selection problem. Initially, each node is assigned a layer, which is equal to its level in the graph. The following steps of the parent selection algorithm are focused on discerning nodes that are closer to the sink and share a connectivity edge with the child node. With each step an attempt is made to find a better solution and when no solution can be found at all, the initial layer assignment is increased, that is, the child node is placed farther from the sink. In such manner more nodes are eligible to act as the parent of the child node.

3.4 Interval data gathering

A special type of environment where communication with the sink is not always possible, requires a different type of MAC protocol. These cases are typical for certain monitoring environments, where data needs to be stored for a long time until the sink comes into view. It is noted that since the time available to forward data to the sink might be limited, energy consumption is taken into account to a lesser degree. The factor which is considered to be most important is the fast data forwarding.

Environments where the mobile sensor nodes occasionally are within the range of the sink pose specific requirements. The protocol proposed in [65] is specifically designed for such environment. Mobile sensors are considered to sense the environment and store all measurement data in nonvolatile memory. Once the sensor comes within range of the sink, it will try to forward all its data as fast as possible. Since the time required to upload the data to the sink is relatively short compared to the data measurement time, energy efficiency is not considered. Two channel access periods are defined within a cycle, a contention period and a transmission period. Both periods are designated by a beacon transmission from the sink, indicating the start and end of the period. Following the contention period start message, nodes contend for the medium by sending RTS messages. Nodes that manage a successful RTS transmission are registered at the sink as members of the transmission group. The sink determines suitable members based on the received signal strength. The transmission period start message includes the schedule of the accepted members. Note that the transmission period is composed of a fixed number of time slots. When more slots are available than the number of nodes, a single node is assigned multiple slots. A Block Ack mechanism is included in the beacon message of the sink announcing the end of the transmission period. Due to the suboptimal performance of the protocol in a sparse environment, a variation is proposed, where only a single node is granted access during the transmission period. As a result, the number of contention period advertisements can be significantly reduced. The node that was granted access uploads all its data to the sink before a new contention period is announced.

The work in [66] targets the optimization of the energy efficiency, data buffering and data throughput, while operating in a network environment where transmissions are only periodically possible. Due to such environment, the network is forced to operate in two modes. During the data gathering period, nodes accumulate data and store it into equally sized data packets. The packets are transmitted towards the sink in the data forwarding phase. The routing tree, which determines the next hop nodes toward the sink, is periodically generated by the sink. The proposed protocol is TDMA based, which allows for a precise control of the transceiver, thereby reducing the idle listening. The tree construction process therefore includes a synchronization activity and the assignment of a unique time slot to each of the n nodes. The sequence of these n slot is called a round. Note that spatial reuse of the slots is not supported. It is claimed that whereas spatial reuse allows a faster collection of the data, the protocol becomes more complex, requires more information and control packet exchanges, and consumes therefore more energy, while the number of transmissions remains the same. When all nodes are assigned only a single slot, each of the parent nodes that have two or more children experience a buffer expansion since the node can send just a single packet at each round while it receives from multiple children. Moreover, nodes that have forwarded all the stored data have no more data to send, while they are still assigned an entire time slot. Such unused slots reduce the effective bandwidth usage of the network. Both issues are considered in the proposed protocol, the buffer usage can be restricted by means of additional information contained in the acknowledgement packet, while the throughput efficiency can be improved by assigning unused slots to the parents. By including slot information in each data packet and a flag whether this is the last packet a node will send, the parent is up to date regarding the slot usage of its child and is able to use some of the free slots of the child. Every second slot that is forwarded by the child is used by the parent. The remaining slots are forwarded to the parent of the parent in order to ensure a more equal distribution of the slots and data forwarding.

3.5 Wireless Sensor Network deployments

While the discussed protocols show interesting research results, this section shows that real-life deployments are not about having a fancy network structure. The main goal is to provide a stable network operation, which allows an autonomous operation, that is, without intervention, for as long as possible. The work in [67] also made a similar observation, where it is said that WSN deployments are often small scale, single hop, static and have a predefined network topology. A study of the available literature showed that the proposed protocols in the literature are often too complex to be used in real life. Moreover, most of the works are not implemented in one of the known operating systems, such as TinyOS or Contiki, making it hard to derive an implementation from the often high level description. This shows that there is a discrepancy between the objectives and assumptions of how a sensor network should behave or look in most MAC protocol designs, when compared to real-life deployments. For example, many papers state that a sensor network is a dense network, which should be capable to operate autonomously, without any fixed topology. In the examples shown below, there is often a sparse network, where the topology of the sensor nodes is optimized in favor of the stable operation of the network.

Habitat monitoring of the usage pattern of nesting burrows is the goal of the deployed WSN in [68][69]. The target is to investigate the alteration between the breeding and feeding role of the Leach's Storm Petrel during a 72 hour cycle. On top of that, the environmental conditions of the burrow during the seven month lasting breeding season is investigated. In the deployment, all sensors form a multi-hop network, based on a tree structure, in which they forward their data to a single gateway. The leaf nodes start sending their data to their parents, after which the following level is activated to send its data to the next level until all data is collected by the gateway.

A Wireless Sensor Network deployment where the habitat monitoring also targeted the monitoring of burrows and seabirds is described in [70]. The targeted birds are Manx Shearwaters, which are being followed by miniature GPS logging devices. The downside of this method is the manual burrow inspection every 20 to 30 minutes to recapture the tracked birds. Thanks to the WSN deployment, researchers have an almost immediate access to arrival or departure information. Moreover, besides the requested information, environmental data regarding temperature and humidity inside and outside of the burrows is captured, providing a better understanding of the ecosystem. The network consists of a small number of sensor nodes placed next to the monitored burrows, a solar powered base station and a mainland server. The sensors are configured in a star topology with the base station acting as sink. The base station ensures synchronization between the sensors and controls the TDMA schedule the sensors should operate in. Once a day all data is forwarded from the base station to the mainland server. The arrival or departure of the birds was detected by two Passive Infrared (PIR) sensors, which activated the RFID reader for five seconds in order to conserve energy, since the RFID reader consumes a lot of power. Both the transmission software and the setup of the sensors needed regularly some adjustments due to external factors, such as light conditions influencing the sensors and juvenile birds. The research also indicated the requirement of a modular system which could easily be replaced or updated, since working in harsh whether circumstances or under the light of a flashlight during the night makes detailed inspection of the system difficult, such as recognizing the color code of the wires.

The work in [71] describes the deployment of a Wireless Sensor Network to collect data for geophysical studies, that is, the study of active volcanoes. Both seismic and infrasonic information is gathered by the network. Typical data collection tools require a car battery as power source and are difficult to move. A sensor network would therefore present a significant improvement. Since the relevant information is situated before and after an event, such as an eruption, earthquakes or tremor activity, the sensor nodes continuously gather sensor information and store the data in a circular buffer. The recently collected data is investigated at each individual node to detect an event. When an event is detected, the node sends a trigger message to the base station. If a sufficient number of nodes reported the event, the base station instructs all nodes to forward the gathered sensor data in a round robin fashion. The communication methods are based on methods available in TinyOS, while the synchronization protocol could be easily integrated in the code base. The synchronization is required to be able to correlate the individual measurements. The experiment resulted in a data retrieval during 61% of the time. Due to the remote location, it was impossible to power the logging laptop 24 hours a day. Moreover, due to a software component failure, all nodes needed a manual reprogramming on site, which resulted in an outage of three days.

3.6 Conclusion

This chapter provided background information on Wireless Sensor networks by elaborating on the most notorious MAC protocols and their improvements within a limited application area. The application domain for WSNs, and therefore also the research performed in the context of WSNs, is widely varying. The popularity for this type of network is a result of the low cost and small form factor of the devices, and their added value when organized in a network.

Most sensor devices are destined to operate on batteries in remote areas where the replacement of batteries is difficult, impractical or discouraged. Therefore, the primary concern of most MAC protocols designed for WSNs is the conservation of energy, where an attempt is made to reduce the waste of energy to a minimum. Sources of energy waste are considered to include overhearing, idle listening, collisions and control packet overhead, of which idle listening is assumed to be the major source of energy waste. The resolution is different for each MAC protocol, some make use of a duty cycle, thereby allowing nodes to enter a low power state periodically. Others employ a method called Low Power Listening, where the medium is sensed for only a short time to determine whether the node should start listening. Most protocols that employ these techniques are contention based protocols. A different class of MAC protocols are the scheduled access protocols, where the access to the medium is meticulously controlled, such that the elimination of idle listening is achieved implicitly.

The different TDMA based MAC protocols for WSNs differentiate especially in their slot allocation approaches, such as a distributed or centralized slot assignment, an assignment based on geographical location, cluster based allocation, etc. A considerable number of works consider the 2-hop distance coloring method as a means to determine a slot allocation where no collisions occur in the two-hop neighborhood of the transmitter. Several works also attempt to find an optimal slot assignment such that a maximal throughput is achieved in an interference free slot allocation. Since this problem is found to be NP-complete, several heuristics are proposed to find one of the possible solutions. Some works consider the priority of packets, such as for example in emergency situations. Other works allow the coexistence of a scheduled medium access and a contention based access in order to improve the performance or stability of the network. Note that most slot allocation methods assume a node suffices with a single slot. Such assumption may lead to a bandwidth deficiency, since the relay nodes need to send both their own data as the data received from other nodes. Moreover, this type of allocation does not take into account a possible difference in the traffic demands of the nodes. Only a few works really consider a heterogeneous network in which sensors might have different traffic demands.

Some special MAC protocols, which can be either contention or scheduled based, focus on the forwarding of data over several hops per cycle. Another set of specific MAC protocols is focused on networks where data needs to be sampled and stored for a certain time since no path towards the sink is available. As soon as a path towards the sink is found, the data is uploaded as fast as possible. For such protocols, the energy consumption is of minor importance when compared to the throughput.

In a last section a number of sensor deployments are discussed, from which it is clear that simple topologies are used with a limited number of hops, often predefined at deployment time. Most of the encountered issues involved unanticipated practical concerns, such as animals disturbing the measurement setup, software errors, etc.

CHAPTER 4

Time synchronization in Wireless Sensor Networks

4.1 Introduction

This chapter discusses the theory and practice of time synchronization in Wireless Sensor Networks (WSNs). It is an often used method in Wireless Sensor Networks and is in many cases even required. The contribution of this work can be found starting from Section 4.5, where an analysis is done of the time critical hardware elements, related protocols are compared using this hardware information in the next Section and a novel synchronizaton method is proposed in Section 4.7. A significant number of works consider a scheduled transmission and reception pattern, that is, TDMA, which usually requires at least a coarse synchronization. Besides the need for frame and slot synchronization, a sensor network might also have the need to correlate the sensor data from different nodes, such as in [71]. In such cases, all nodes in the network should use the same time reference. Note that synchronization does not imply the linking of the global time of a network to an external standard of time. However, if it is required, it can be achieved by connecting the sink to an external time server, such as a GPS device. The objective of the discussed synchronization methods in this chapter is to enable a scheduled transmission and reception. Synchronization to introduce a global timescale to correlate measurement timestamps is not the targeted application, although implicitly this is also made possible.

The tracking of time is realized by incrementing a counter at a periodic rate. As previously mentioned, the rate at which this happens does not need to be equal to the periodicity of an external time reference. Such counter, also called a timer, is usually initialized to zero at startup. In other words, each sensor begins incrementing its time counter, starting from zero, as soon as it has been activated, i.e., when it is switched to the 'on' state. As a result, if not all devices have been activated exactly at the same time, they will have a different time counter value, which is the reason that synchronization is required to make sure all nodes have equal counter values. This initial synchronization can be obtained by using one of the existing synchronization mechanisms that are discussed in Section 4.4. Any type of synchronization is usually based on the exchange of information, either by sending explicit information, such as timestamps, or by means of indirect information, such as the reception time of a packet. Unfortunately, such initial synchronization does not ensure that nodes stay synchronized indefinitely. The clocks of the devices, which determine the rate at which the timers are incremented, are designed to have a very high precision. However, two clocks will not have exactly the same frequency. Due to the inequality in the rate at which these timers are updated, the sensors are required to perform a synchronization operation on a periodic base.

The phenomenon of diverging timer values is called clock drift. The measure of clock drift can be expressed by two different metrics, the clock skew, that is, the difference in frequency of the two clocks, and the clock offset, that is, the instantaneous clock difference. The sensor node clocks exhibit a certain offset relative to the other clocks from the start due to the different startup times, i.e., they experience a clock offset. On the other hand, the clock skew is a result of the deviating clock rates of the sensor nodes. In order to understand the dynamics of this phenomenon and counteract its impact by means of an efficient algorithm, sufficient knowledge about its origin is required.

Clock drift is a physical effect that manifests itself due to a gap between the theoretical and realistic operation of hardware. The production process of hardware is precise, but is not able to produce perfect components. All hardware is said to operate within certain tolerances. In order to determine the components that could influence the clock drift of sensor nodes, this paragraph enumerates the different hardware components a sensor node is composed of. A typical sensor node consists of different components, usually an MCU (microcontroller), a wireless transceiver, either integrated into the MCU or as an external unit, an antenna system, a power supply regulator and sensors. Both the MCU and the transceiver require at least a single periodic signal as input in order to operate, some components even provide the option of having several clock inputs. The clock signal is used to derive an internal master clock signal, which operates at a stable frequency and determines the operational speed of the component, that is, the rate at which instructions are executed is derived from this clock. A microcontroller contains several peripheral components that require a clock signal to control their operation. Examples of such components comprise USART, SPI and I2C amongst others. Since not every peripheral operates at the same frequency, the master clock is often used to derive several clock signals, amongst which a peripheral clock that can be downscaled through prescalers for each peripheral component. The timers, which use one of the internally defined clock signals as reference input, are implemented as counters that increment their counter value at every rising or falling edge of the clock source, which usually depends on the configuration of the MCU. By means of configuring a preset value to which the timer value should be compared with, the system is able to generate interrupts at specific time instants. As such, one of the available timers can be used as a time reference for the sensor node. A mismatch between the timer values of different nodes, even when they had the same value at the start, can be attributed to the clock drift.

The precision of the clock signal is affected by for example variations in temperature of the MCU, however, the major part that attributes to this phenomenon can be traced back to the periodic signal which is used as input for the MCU. That signal is usually generated by an external crystal, external oscillator or can even be generated internally in the MCU, which is called an internal clock oscillator. The latter is usually based on an RC oscillator, providing a low cost solution where no extra external components are required. The downside of this type of oscillator is its usual low precision, even though its frequency is likely to be tunable. The resistor (R) value can be digitally tuned while the capacitor (C) value stays constant. The reason for this is discussed in more detail later in this chapter, but basically, constructing a resistor in silicon requires less space than a capacitor and is therefore cheaper.

In order to understand the dynamics of the imprecision of the clock source, both the quartz crystal and the RC oscillator is investigated. Note that there exists many variations to the RC oscillator, some of which are described in chapter 15 of [72]. However, to go into detail to the specifics of the operation of all these RC oscillators, would be out of the scope of this chapter. Moreover, the physics behind the deviation of the frequency is similar for all types of RC oscillators, that is, the precision of the resistor and the capacitor.

Both the external crystal and external oscillator can be expected to be operating with a precision of 40 ppm (parts per million). High precision, and therefore also more expensive components, achieve an even better precision. However, since a sensor node is expected to be cheap it is unlikely to find such high precision components in a sensor node. The internal oscillator is dependent on the typical precision of both the R and C components, which is around 1% for high precision configurations. Very high precision resistors can be expected to have a precision of around 0.1%. The following paragraphs first discuss the operation and environmental influences of the quartz crystal, after which one of the RC oscillators is discussed.

A quartz crystal oscillator is comprised of a very thin piece of quartz with two parallel metalized surfaces for the electrical contacts [73]. Thanks to the piezo-electric effect the quartz crystal changes shape when a voltage source is applied. A mechanical force is produced by applying an electrical charge. Likewise, an electrical charge is generated when a mechanical force is applied. This effect produces mechanical vibrations, that is, oscillations at a certain frequency, determined by the size and thickness of the quartz. Although a quartz crystal is considered to be very stable, it exhibits deviations from its proclaimed frequency due to several factors [74].

No quartz crystal oscillator is perfect, there can be impurities in crystal growth, imprecisions in the cut process or uneven thickness. This is called the frequency tolerance and is expressed in a deviation of parts per million (ppm). Typical values range from 5 to 100 ppm, where 20 to 30 ppm is an often used value. Note that even a frequency deviation of just 10 ppm would cause a clock error of about one second per day. The crystal tolerance is usually specified at a reference temperature of 25 °C[75]. Lower and higher temperatures influence the frequency deviation as depicted in Figure 4.1. Therefore, the emitted heat by the MCU or the environment in which the sensor is placed could result in a larger deviation from the specified frequency.

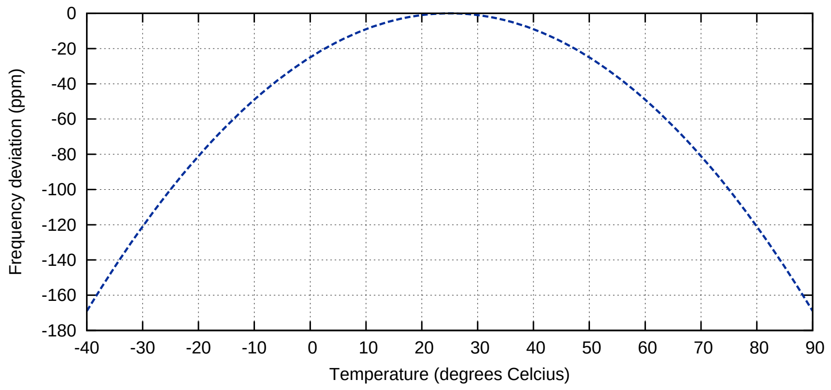


Figure 4.1: *Typical Crystal Frequency deviation due to temperature. (Ref: Adapted from the datasheet of a 32.768kHz crystal from EuroQuartz[75].)*

A third factor that influences the actual frequency of crystals is aging, to which crystals are subjected due to their electromechanical character. There exists no simple prediction model that could dictate the aging of a crystal, although it is known that crystals age the most during the first hours of operation. Very frequency sensitive applications are therefore equipped with aged crystals, which exhibit a lower aging rating in ppm/year.

A commonly employed oscillator that works with crystals is the Pierce oscillator, which is depicted on the left side of Figure 4.3. Such oscillator is also used in the MSP430 devices and requires two external capacitors as load for the crystal [76]. The capacitors used for these purposes are often NP0 capacitor types, also known as temperature compensating capacitors. They are class 1 ceramic capacitors and are known for their stability regarding to temperature (30 ppm/°C). They also have the lowest volumetric efficiency among ceramic capacitors and are therefore only available in very low capacitance values, in the range of 0.1pF till 10nF, with a tolerance between 1% and 10%. Note that the PCB (Printed Circuit Board) layout, the width of the traces, number of through holes, etc. also introduce some parasitic capacitance, which needs to be taken into account. It appears that despite the interesting tolerance qualities of a crystal, the resulting oscillating frequency might be having a larger frequency deviation due to the higher tolerances of the capacitors, not to mention the parasitic capacitances, which can introduce an additional 2 to 3 pF. However, thanks to the properties of a crystal and the design of the Pierce oscillator, the influence of both the load capacitors and the parasitic capacitances are reduced.

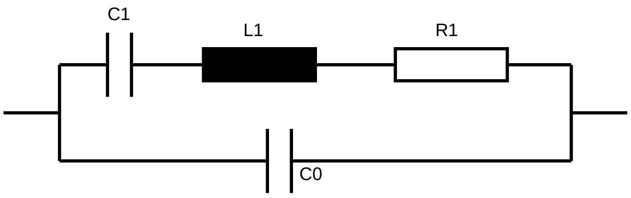


Figure 4.2: *Equivalent electrical circuit of a crystal. (Ref: Adapted from Texas Instruments Application Report SLAA322B, revised April 2009, [76], Figure 2, page 2.)*

An important piece of information is embedded in the electrical equivalent circuit of a crystal, which is depicted in Figure 4.2. It shows that the behavior of a crystal can be modeled as the motional capacitance C_1 in series with an inductance and a series resistance. In parallel to those components, is the shunt capacitance C_1. Based on this information, the sensitivity towards any deviations of the capacitance value in the Pierce oscillator circuit can be calculated [77]. In order to make the calculations easier, first the load capacitance, which is defined as the amount of capacitance measured across the crystal terminals on the PCB, is calculated by means of Equation 4.1. C_{in} is the input capacitance of the inverter and C_{out} is the output capacitance of the inverter gate. The pcb strays are represented by C_{stray}.

$$C_L = \frac{(C_{in} + C_1)(C_2 + C_{out})}{C_{in} + C_1 + C_2 + C_{out}} + C_{stray} \tag{4.1}$$

The sensitivity of the frequency towards the changes in the load capacitance can therefore be calculated by means of the Trim Sensitivity equation:

$$S = \frac{C_1}{2(C_0 + C_L)^2} \times 10^{-6} (ppm/pF) \tag{4.2}$$

In this equation, C_1 is the motional capacitance of the crystal, C_0 is the shunt capacitance of a crystal and C_L is the load capacitance. It is clear that the impact of the load capacitance is diminished by the entire circuitry and therefore does not nullify the excellent properties of the crystal. However, at design time, the characteristics of all elements should be taken into account in order to attain a certain frequency precision.

While crystal oscillators are mainly influenced by the precision of the crystal which is the key component ensuring an oscillation, RC oscillators are characterized by the use of a combination of a resistor (R) and a capacitor (C) to ensure appropriate

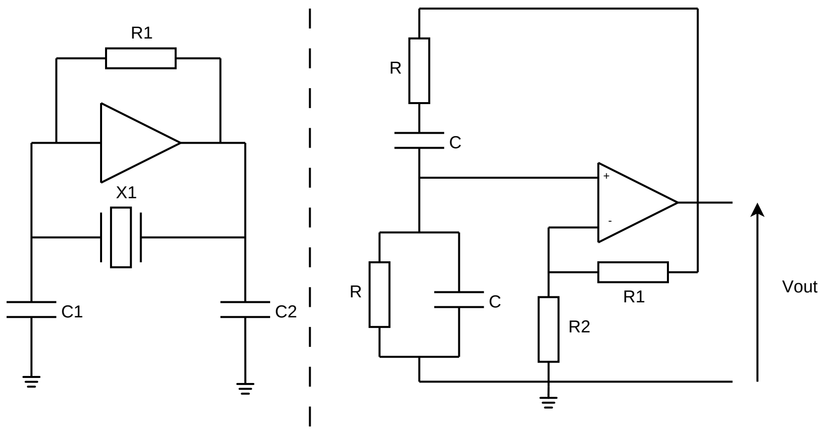


Figure 4.3: *Pierce oscillator (left) and RC oscillator (right). (Ref: Adapted from web tutorial on Quartz Crystal Oscillators,* `http://www.electronics-tutorials.ws/oscillator/crystal.html`*, [73].)*

oscillation. While many types of RC oscillation circuits are available, only the Wien Bridge Oscillator circuit is discussed which is a commonly known circuit. The Wien Bridge oscillator, depicted at the right in Figure 4.3, is known for its good stability at its resonant frequency, low distortion and is very easy to tune. It makes use of a series RC circuit, acting as a high pass filter, connected to a parallel RC circuit, which acts as a low pass filer. Such circuitry results in a band pass filter with a high Q factor at the selected frequency. Moreover, the RC circuits ensure a phase shift, except at the resonance frequency. Therefore, a positive feedback will occur only at the resonance frequency, which is one of the requirements for oscillation. The resonance frequency is determined by the R and C values and is independent of R1 or R2, which determine the voltage gain of the amplifier. The frequency can be expressed as:

$$f_r = \frac{1}{2\Pi RC} \tag{4.3}$$

The precision of the RC oscillator is therefore highly dependent on the precision of the resistor and capacitor values. Like the crystal, the precision of the resistor can also be classified by its tolerance and temperature coefficient. Typical high precision tolerance values are 0.1% (that is 1000 ppm), while deviations towards temperature tend to be around 400 ppm/°C. The deviation towards the proclaimed value of capacitors can even be as poor as 10% [78], however, by calibrating either R or C, a precise frequency can be generated. Due to physical constraints and the manufacturing process, the most cost effective method is to trim the R value, since it is much smaller than a capacitor. In order to digitally control the R value, a resistor array can be made available, where a series of resistors have NMOS switches parallel to each resistor, which allows each resistor to be shorted, thereby reducing the total resistance.

To summarize, the clock drift experienced on sensor nodes is caused by the imprecision of the production process, for both crystals as RC oscillators. However, aging and environmental circumstances, such as change in temperature, can cause fluctuations in the imprecision of the clock drift. It should be noted that designs that consider operation without the aid of a high precision crystal, that is, solely relying on the internal RC clock generator, are far less precise and therefore require a more frequent synchronization message exchange.

In the next section an overview is provided of the theoretical background of time synchronization, which is employed in practice in Section 4.3 for a general distributed network. The related work which is specific for time synchronization in a WSN is presented in Section 4.4, where the synchronization protocols are classified according to the transmit or receive role of the participating nodes. Some of the hardware architectures that can be used in WSNs are discussed in Section 4.5, while the oscillator inaccuracies and the effect of the hardware architecture is discussed in Section 4.6. Based on the observations of the previous section, a new algorithm is proposed in the next section, which is evaluated on different types of hardware in Section 4.8. Concluding remarks close this chapter.

categorized according to the communication method and flow o fthe messages

4.2 Principles and Concepts of Time Synchronization

Since the clocks are imperfect, they exhibit a deviation from their desired frequency, which is called clock drift. In order to define the problem, Kopetz [14][44] suggests the existence of a perfect reference clock with infinite granularity and defines the rate at which a clock oscillates and generates events, to be a microtick. Figure 4.4 depicts a perfect clock (the solid blue line) and the area which contains possible values of the imperfect realistic clock when both reference clock and realistic clock were synchronized at the start. The drift of a clock between microtick i and microtick $i+1$ is modeled as the duration of a granule of clock k, measured with the reference clock z and dividing it by the nominal number n^k of reference clock microticks in a granule as depicted in Equation 4.4.

$$drift_i^k = \frac{z(microtick_{i+1}^k) - z(microtick_i^k)}{n^k} \tag{4.4}$$

The drift rate ρ can be defined as: $\rho = \left|drift_i^k - 1\right|$. A perfect clock has a drift rate of 0.

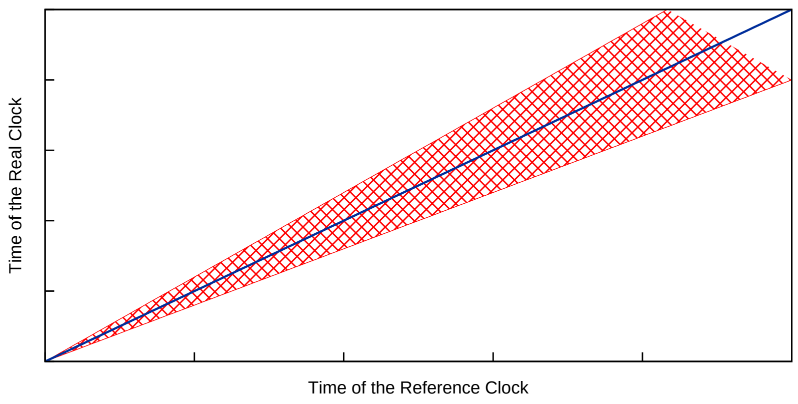


Figure 4.4: *Time drift of a clock. (Ref: Adapted from "Real-Time Systems, 2nd edition", H. Kopetz, Springer, 2011 [44], Figure 3.1, page 55.)*

Another frequently used notation is cited in [79], where it is claimed that if there exists some constant ρ, such that for a clock value C Equation 4.5 holds, the clock is assumed to be working according to its specifications. The constant ρ is known to be the maximum drift rate.

$$1 - \rho \leq \frac{dC}{dt} \leq 1 + \rho \tag{4.5}$$

The maximum clock offset between clocks can then be formulated as $2\rho\mathit{\Delta}t$. Therefore, in order to guarantee a maximum clock offset of δ, the two clocks should need to be synchronized every $\delta/2\rho$ seconds. Note that although the equations don't take the variability of the drift into account, the model is correct for sufficiently small time intervals.

When discussing synchronization methods, the eventual goal needs to be considered. The distinction needs to be made between internal synchronization, that is, ensuring an alignment of the clock's microticks within the network, and external synchronization, that is linking the time view of the network to an external time standard [14][44]. For the latter, it is necessary that at least the gateway has access to a timeserver, for example a connection to a GPS receiver.

The modus operandi of the synchronization method can be either centralized, or distributed. The centralized method makes use of a central master, which periodically sends the value of its clock to all other nodes, the client nodes. The client records the time of arrival of such timestamp messages. The difference between the master's time, contained in the synchronization message, and the recorded client's time-stamp of message arrival, corrected by the known latency of the message transport, is a measure of the deviation of the clock of the master from the clock of the client. The client then corrects its clock by this deviation to bring it into agreement with the master's clock. The synchronization error is determined by the difference between the fastest and slowest message transmission to the client nodes of the ensemble, i.e., the latency jitter.

In the distributed method, every node acquires knowledge about the state of the global time counters in all the other nodes by the exchange of messages among the nodes. By analyzing the collected information, a correction value for the local global time counter can be calculated. The local time counter of the node is adjusted by the calculated correction value.

Note also the difference between state correction and rate correction. Correcting the state of the clock ensures that the two clocks are nearly synchronized at the time of synchronization. However, every $\delta/2\rho$ seconds, with δ being the maximum allowed clock offset, a new synchronization needs to be performed, since the clocks drifted apart again. Note that state correction should always correct the clock in the future. An excellent example is given in [14], where at New Years Eve at 12:00 midnight the time was corrected a couple of seconds backwards. The next transition to a full minute resulted therefore also in the transition to a new day and instead of January 1, it had become January 2. Rate correction, on the other hand, tries to correct the rate at which the clock is incrementing. By employing this method, the skew rate could be lowered and therefore, synchronization does not need to be performed that frequently, or when performed at the same interval, the clock offset will not be as large.

4.3 Synchronization in Distributed Systems

In [80] several synchronization methods are discussed for distributed systems. Although they are unsuitable for WSNs, they form the base of contemporary synchronization protocols for WSNs. The discussed methods do not consider practical issues, they focus solely on the methodology and discuss the error margin from possible sources of error.

The work in [81], called Cristians algorithm, proposed an external clock synchronization at the application software level using a probabilistic technique. The proposed algorithm is depicted in Figure 4.5. At time T1, a node queries the state of the clock at another node by a query-reply transaction. The message arrives at time T2 at the receiver of the query, which replies immediately to the requester with its clock value. At time T3, the requester obtains the requested information. The duration of the request-reply transaction is measured by the sender. The received time value is corrected by the synchronization message delay that is assumed to be half the round-trip delay of the query-reply transaction (assuming that the delay distribution is the same in both directions). This methodology is straightforward, however, its accuracy depends on the precision of the time insertion at the transmitter and the processing time of the receiver. Only an estimate of the transmission delay is

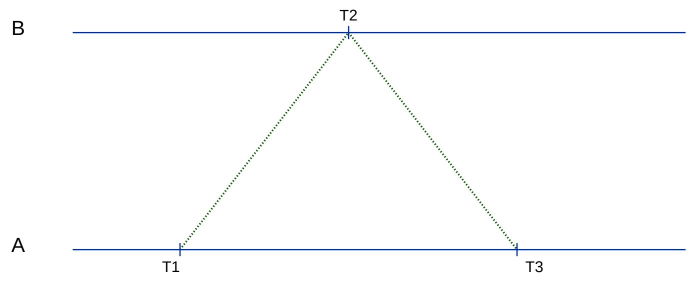


Figure 4.5: *Cristians algorithm, message flow and timing*

taken into account to calculate the clock offset. By running multiple iterations of the algorithm a more reliable result can be obtained by either selecting the results with the lowest round trip time or by averaging the measurements.

The principle of Cristians algorithm is also used in [82], where a coordinator computer acts as master. The other computers, which are called slaves, are regularly requested to send their current clock value. By recording the corresponding round trip time, the master estimates, by means of Cristians algorithm, the clock value of all computers according to its own local clock. The fault tolerant average of all estimated clock values is computed by removing the k largest and k smallest values from the list and calculating the average of the rest of values. The amount by which each of the slaves needs to be adjusted is sent to them, which eliminates the uncertainty caused when sending an absolute adjusted time.

A most notable network synchronization protocol was described in [83][84], called NTP. The algorithm defines three modes to synchronize nodes, of which symmetric mode ensures the most precise synchronization. The algorithm is an improvement of Cristians algorithm, since it only takes into consideration the time required to transfer the messages. The protocol eliminates the dependency on the time between T2 and T3 in Figure 4.6. This protocol is later referred to as the classical two-way message exchange method.

Every message that is sent contains timestamps of recent message events, such as the local time when a previous NTP message was sent and received, but also the local time when the current message was sent is included. The clock offset and the total transmission time, called delay, can be estimated from the information gathered by collecting a pair of messages. Assuming the actual clock offset at B relative to A is equal to *o* and if the actual transmission durations at times T1 and T3 are respectively t and t', then Equation 4.6 is true.

$$T_2 = T_1 + t + o \qquad \textit{and} \qquad T_4 = T_3 + t' - o \tag{4.6}$$

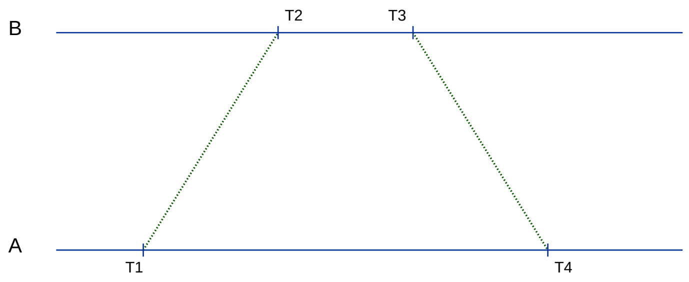


Figure 4.6: *NTP, message flow and timing. (Ref: Adapted from "Distributed Systems: concepts and design", Coulouris et al., Pearson Educated Limited, 2001 [80], Figure 10.4, page 395.)*

By combining both equations, an estimation can be derived for the experienced delay, that is the total transmission time, as in Equation 4.7 and the clock offset, as in Equation 4.8.

$$d_i = t + t' = T_2 - T_1 + T_4 - T_3 \tag{4.7}$$

$$o = o_i + (t' - t)/2, \qquad \textit{where } o_i = (T_2 - T_1 + T_3 - T_4)/2 \tag{4.8}$$

Since t *and* $t' > 0$, the following statement is also true: $o_i - d_i/2 \leq o \leq o_i + d_i/2$. From this equation can be deduced that o_i is an estimate of the offset and d_i is a measure of the accuracy of this estimate. By means of multiple iterations of the algorithm, the value o_j with the lowest d_i is used as actual clock offset.

In networks, the transmission and reception timing of a message is influenced by many factors, such as the scheduler, the transmission queues, concurrent message transmissions and medium access delays. Since the precision of the synchronization is in a great deal affected by the jitter of the time messages that contain the current clock values, probabilistic protocols were designed to reduce the effect of the jitter. The protocol designed in [85] is a probabilistic clock synchronization protocol for wired networks. A single node acts as a master, while the rest of the nodes act as slaves. The slaves are synchronized to the master clock by sending its local clock in a series of N messages, each of which contains the local time at which the message was sent. The receiver estimates the current time by means of the time of arrival and an estimation on the message delay.

A somewhat different approach to ensure ordering of events in a network is proposed in [86]. The proposed method deviates from other previously discussed methods, since it does not provide a method for time synchronization, but allows for causal ordering of events. Before every event that occurs at a node, the clock is incremented to $L_i := L_i + 1$. Each message that is transmitted carries the current logical clock value, L_i. The maximum clock value is chosen between the received clock value and the local clock value upon reception of a message. While this type of approach may be interesting in some cases, it is unlikely to find its application in sensor networks.

4.4 Synchronization in WSNs

Time synchronization is of major importance in WSNs for a multitude of reasons, some among which are ordering or correlation of the recorded sensor values in order to detect duplicate recordings, agreement protocols, TDMA scheduling, power saving and object tracking. Although a considerable number of protocols have been designed for distributed networks, they can not be employed for a Wireless Sensor Network without modifications. WSNs need to comply to specific requirements and by extension also their related protocols. The most referred to requirements [87] can be described as energy efficiency, scalability, robustness and ad hoc deployment. Since the synchronization protocol should consider the limited amount of energy, it should try to exchange as few as possible messages. Moreover, the synchronization should be fast and trivial, no complex algorithms or calculations should be required. Since WSNs can become rather large, scalability should also be considered. A synchronization protocol should therefore avoid time negotiations with the entire network, this would lead to an unmaintainable situation in a rapidly growing environment. Since sensors could break down, the synchronization should be flexible and either not depend on a single sensor node, or be able to reconstruct the network functionality. Those specific requirements make existing synchronization protocols for distributed networks unsuitable for WSNs. In [87] the argument is made that NTP is not an appropriate protocol for WSNs due to the frequent re-adjusting of the system clock and the lack of limits on the number of transmissions. However, the core idea of NTP, that is the two-way message exchange is a qualified methodology for use in WSNs.

An often used argument is that there is no predefined topology or structure in WSNs, since sensor networks do not have an external infrastructure, or a hierarchy rooted at a single node. Synchronization sourced from the root node, ending at the leaf nodes would result in a large difference in sync error between root and leaf. This statement should be reconsidered since a sensor network is in most cases composed of sensor nodes and one or multiple sinks. Perhaps the sink is not available all the time, but it is present, since otherwise, the sensor network is an isolated entity with no means to transmit the gathered data. Moreover, there is no requirement for the lowest leaf to be perfectly synchronized with the root in case the goal of the synchronization is to enable transmission scheduling. The network is able to sustain itself as long as there is a local agreement, ensuring there are no collisions. When considering real life deployments, a sensor network needs to be deployed with care and in most cases N tier hierarchies are avoided as much as possible, since this could result in an unfair battery usage due to the funneling effect on the traffic load. It is better if the maintenance is able to predict when batteries should be changed and easier when all batteries need to be changed at once.

The design of a considerable number of protocols takes into account the differentiation between the sources of time synchronization errors due to the transmission of messages, which is composed by Kopetz and Schwabl [88]. The work makes the distinction between four different components that attribute to synchronization errors. The first is the '*Send Time*', which is the time spent at the sender to compose the message. Sources of variable delay could be context switches, system calls and transfer of the message from memory to the radio. The second component is called the '*Access Time*', which takes into consideration the time to access the wireless medium. Clear channel assessment, RTS/CTS and backoff schemes are prime sources of variability. The third component, '*Propagation Time*', is the time required to transmit the message over the medium to the receiver. In the context of a wide area network, the variability can accumulate to a large value. In WSNs, this parameter is considered the physical propagation time, which depends on the physical distance between nodes. The last component, '*Receive Time*', accounts for the required processing of the received packets. By timestamping the reception time at an as low as possible level in the communication stack, possible sources of variability are avoided.

The synchronization protocols that are to be described in the following subsections can be categorized according to the communication method and flow of the messages. Protocols that receive a message and synchronize against the other receivers are considered receiver-receiver based protocols. Protocols that perform some negotiation between sender and receiver and where the receivers synchronize to the sender are considered to be sender-receiver based protocols. The last category is where senders transmit clock information and receivers try to estimate the clock offset based on knowledge of the hardware, or by means of iterations.

4.4.1 Receiver-Receiver based synchronization

The protocol that is probably the most discussed time synchronization protocol for WSNs is Reference Broadcast Synchronization (RBS) [54]. Like [85], it is a probabilistic protocol. Instead of synchronizing the sender with the receiver, RBS provides synchronization between receivers. A sender node broadcasts beacons at random times. The beacons do not need to contain any specific information, since the reception of the beacon itself is the information. All receivers record the time of reception of such beacon and exchange this information amongst all receivers, which enables them to make an estimation on their relative skew and offset. By employing a least-squares linear regression on a limited set of timestamps of the received beacons, the accuracy can even be improved. Interesting is the elimination of sources of non-determinism, such as '*Send Time*' and '*Access Time*', according to the classification in [88]. The only remaining sources are the '*Propagation Time*' and '*Receive Time*'of which the impact is limited since the '*Propagation Time*' is small considering the radio wave propagation speed, and the '*Receive Time*' can be reduced by employing a hardware triggered capturing of the current time.

A major disadvantage of RBS [54] is the lack of coordination when the network is larger than just the broadcast region. Moreover, when the functionality of sender node is not exchanged with other nodes, a scheduled transmission and reception is not possible with the sender node. A rotation of the sender functionality would alleviate this issue, however would incur a significant number of synchronization traffic. [89] addresses both issues by proposing a protocol which is based on RBS and the probabilistic character as in [81]. The parent node broadcasts n beacon packets with a fixed transmission interval. The children nodes each record the time of arrival and determine the line which fits these timestamps by means of linear regression. Each child node sends the slope and a point of this line to the parent node. In order to reduce collisions, a random backoff scheme is used. The parent node then broadcasts this information to all children. Thanks to this information, all children are able to determine their relative drift and skew to each other and the parent node. The advantage of this approach is that the nodes can become synchronized to their parent and furthermore, the parent node is always the center of operations. There is no risk that the children are unable to communicate to each other and therefore are unable to exchange timing information. In a multihop scenario, the protocol proposes a step wise synchronization. First the nodes in the direct neighborhood are synchronized. Afterwards, the one hop neighbors are synchronized by the already synchronized nodes. This process continues until all nodes are synchronized in the network.

A set of alternative approaches to use RBS in larger types of networks is proposed in [90] and [91]. The first approach is to organize the network in clusters, of which the cluster heads are amongst themselves synchronized. Such synchronization is achieved by an estimation of the total round trip time through all cluster heads and calculating the average time difference per cluster head by dividing the estimation through their number. The nodes within the cluster are synchronized to each other by means of RBS if all cluster members are within hearing distance of the cluster head. The second proposed approach is to use RBS to gather time difference information of all neighbors and take the average as their own clock. Since all nodes perform the same operation, after a number of iterations, the entire network should become synchronized.

Note that thanks to the elimination of the dependency on '*Send Time*' and '*Access Time*' at the sender side, the previously discussed synchronization protocols are suited to be used with a contention based medium access scheme. Especially the variable access time of a such medium access would gravely affect the accuracy and reliability of any other type of synchronization protocol and are therefore unable to use a contention based medium access.

4.4.2 Sender-Receiver based synchronization

One of the earliest works for time synchronization in WSNs is proposed in [92]. The protocol is designed to send the timestamp of the local clock to other devices. Since the clocks of different devices are uncorrelated, the timestamp is first converted into a common time transfer format, UTC. At reception, the UTC time is converted back to the local clock time. The transformations lack some precision, however, a lower and upper bound can be estimated by means of the clock skew, the round trip time of a packet and the idle time that a sync packet is stored to forward again. These estimations determine in which bounds the synchronization error will stay.

Although the work in [92] is interesting, such approach is inappropriate for accurately scheduling the transmissions and receptions. Therefore, most of the works presented in this section refer to the classical two-way message exchange, which was proposed by [84], where the synchronization error is said to be bounded by the message delay.

The protocol proposed in [93], [94] and [95] employs the two-way message exchange in order to gather time information. In order to further increase the accuracy and to compensate for the skew, linear regression is used on those data. Since the procedure could be computing intensive, two different protocols are proposed, Tiny-Sync and Mini-Sync, where the number of data points are reduced to only four points in the former and a more relaxed reduction in the latter. The protocols do not change the local clock, they keep track of the relative clock skew and clock offset to compensate for each of the clocks.

A more deterministic protocol, called TPSN, is a time synchronization protocol proposed in [45], which also makes use of the classical two-way message exchange to synchronizes nodes. In the first phase a virtual tree structure of the network is built, where each node is assigned to a level in the tree by flooding the network with *level discovery packets*. The second phase is initiated by the root node by sending a *time_ sync* packet. The nodes from level 1 (the root is level 0) wait for a random time and then initiate the two-way message exchange with the root node. Since the nodes from level 2 can overhear this communication, they backoff for a random time to make sure all nodes of level 1 are synchronized to the root node and start their own synchronization phase with the level 1 nodes. By taking the timestamp as low as possible in the communication stack, that is in the MAC layer, the authors claim to minimize the uncertainty at the sender. While reducing the uncertainty at the sender side, the protocol introduces a new uncertainty which negatively influences the synchronization accuracy. By allowing nodes to start the synchronization exchange after waiting for a random time, collisions could occur. Even worse would be if the message exchange of a pair of nodes is interrupted by a second pair of nodes, requiring the first pair to wait until the end of the transmission of the message of the second pair. A more convenient method would be to schedule the individual message exchanges, such that no two-way message exchange is interrupted.

A similar protocol, LTS, was proposed in [96], where first a low depth spanning tree is constructed, since with each level, the synchronization error relative to the root node increases. The root node is responsible for initiating a synchronization phase, where a pair-wise synchronization, by means of the two-way message exchange method, is performed along the edges of the spanning tree. The root node initiates a new synchronization round depending on the required accuracy and the depth of the tree.

While the former protocols provide a straightforward method to synchronize the entire network, they unfortunately also require a considerable amount of communication. The proposed protocols in [97] and [98] try to reduce the required amount of communication. The idea of both protocols is similar, that is, a pair of nodes perform a time synchronization through means of a two-way message exchange, while other nodes overhear the exchange and estimate or deduce their own offset and skew. Both protocols rely on the property outlined in [54], where it is said that the reception of a packet is at nearly the same time available at two nodes. The difference is that in [97], there is no assumption regarding network topology and the two-way message exchange is initiated by the child node and an estimation of the skew and offset is made depending on the reception times and data embedded in the communication packages. On the other hand, in [98] a spanning tree is created in a similar manner as in [45] and the message exchange is initiated by the parent node and broadcasted to its children. Only one of the children nodes responds and participates in the message exchange. The parent is able to estimate its relative clock offset towards the child node and broadcasts this information to all its children. Since all children received the initial message around the same time, all children are able to synchronize to the parent node thanks to the clock adjustment information contained in the second broadcasted packet.

4.4.3 Receiver-Only based synchronization

A protocol which could be mistaken to be a derivative of RBS is SLTP [99]. The network is first organized into clusters by means of passive clustering. The cluster head broadcasts at random intervals n time sync messages, containing the local time of the cluster head, towards its cluster members. At reception of those packets, a timestamp of the local clock is taken by each of the members. By means of linear regression the clock skew and offset relative to the cluster head is calculated. The protocol keeps a list of clock skews and offsets relative to the other nodes instead of adjusting its own clock. This protocol requires the timestamp of the cluster head to be embedded in the broadcast packet, thereby failing to eliminate the '*Send Time*' and '*Access Time*' such as in [54].

The protocols presented in this subsection do not attempt to eliminate those sources of uncertainty, however, they try to minimize them. Therefore, the approach taken by the former protocol is not a matter of poor decision making when it is properly executed, such as proposed in [100] and [101]. Both protocols make use of detailed knowledge about the radio hardware.

In [100] a packet containing the timestamp at transmission time of the sender is passed to the receiver. By taking the timestamp as low as possible in the communication stack, more specifically, after the MAC delay and backoff, the uncertain duration of the upper layer processing is eliminated from the time synchronization. The protocol is focused on a contention based access mode, which is known for its uncertain transmission properties.

Another protocol that takes advantage of detailed knowledge of the radio hardware is called DMTS [101]. The protocol is designed such that multiple applications are able to use the hardware time in 'parallel' by checking the timer configuration at every timer interrupt. The time synchronization is done by a master that broadcasts its current time to other nodes. Since this timestamp is set as late as possible, the '*Send Time*' is claimed to be eliminated. By analyzing the behavior of the radio and packet transmissions, the packet delay can be accurately modeled. The receivers measure time of reception by relying on a hardware interrupt which is triggered at the arrival of the packet. The correction that is applied to the local clock of the receivers is the timestamp of the transmitter embedded in the packet plus the experienced packet delay.

A different approach is taken by the protocol proposed in [102], Flooding Time Synchronization Protocol (FTSP), where the sender node transmits an estimate of the global time to the receiver. Timestamps are taken at a level as low as possible in the communication chain, thereby reducing the sources of error. At the reception side, not only timestamps are taken at reception of the packet, but also at every byte boundary in order to reduce the invariance due to the interrupt handling procedure. It is claimed that simply sending a timestamp is not sufficient and therefore linear regression is applied on the timestamps to be able to compensate for the clock skew. The protocol does not build a fixed tree structure, however makes use of an ad hoc structure in order to synchronize the entire network.

Since with every hop the synchronization error towards the root node enlarges, the protocol in [103] is proposed to ensure an evenly distributed synchronization error. A beacon containing a global reference timestamp is broadcast to all neighbors. At reception is not only a timestamp of the time of arrival taken, but also of the first six byte boundaries in order to be able to compensate for the interrupt handling variance. Linear regression is performed on k values to compensate for the drift relative to the root node. The goal of the protocol is to disseminate the global reference timestamp as fast as possible through the network in order to reduce the time between the first and last synchronization.

A more controlled methodology is proposed in [104], the Time-Diffusion synchronization Protocol (TDP), where beacons containing a global time reference timestamp and a deviation parameter are broadcast. The receivers acknowledge the reception of the beacon, thereby giving the master the opportunity to calculate the round trip time. The average of all acknowledgements and their standard deviation is recorded by the master. The neighbors of the master determine which of them will act as leader, to rebroadcast the timestamp, by performing a diffused leader selection. Every diffusion leader adds half of the round trip time to the timestamp and accumulates its round trip time deviation into the deviation which is to be sent. All received timestamp information is eventually gathered into a table by the sensors, accompanied with the accumulated deviation over the number of hops. After a predefined time, all nodes assign a certain weight to the stored timestamps that have been gathered in their time data tables, where the timestamps with a lower associated deviation obtain a higher weight. A new timestamp value is calculated by the summation of the recorded timestamps, according to their weight.

Whereas in TDP a weight is assigned to certain timestamp values to determine its validity and thereby taking into account the values from different sensors, the following two protocols use a consensus based protocol to collect information from different sensors and determine a common clock value. In [105] a consensus based time synchronization protocol is proposed, where beacon packets are broadcast at periodic intervals. By means of a trivial linear equation the skew and offset can be determined. Instead of directly applying the resulting skew and offset, a low pass filter is used, which can be parametrized in order to control the balance between a fast consensus and an error free consensus.

The protocol studied in [106] and [107] is also consensus based, however, only a single timestamp pair is stored. The skew and offset are determined by using a trivial linear equation and is immediately used to correct the clock. Timestamps from other neighbors eventually ensure a global consensus is reached.

The circumstances and parameters of an underwater acoustic sensor network are significantly different from a traditional terrestrial sensor network. Due to the absorbtion properties of water, a traditional electromagnetic field has a very limited propagation. Therefore, acoustic telemetry if an often used underwater communication method. Unfortunately, this type of communication is subject to a lower bandwidth, path loss, noise, multi-path propagation, Doppler spread and high and variable propagation delay [108][109][110]. Whereas in traditional sensor networks the propagation delay is often assumed to be negligible, it becomes an important factor in underwater acoustic communications. Especially the propagation delay, multi-path propagation and the high bit error rate ensure that traditional synchronization protocols, such as RBS [54] can not be used in these types of networks. In order to cope with these limitations and the specific communication environment, a specific method was devised in [111][112]. At first the skew between the nodes is corrected by means of sending broadcast packets containing a global time reference timestamp. By means of linear regression on a set of beacon values do the nodes achieve synchronization of the skew. Afterwards, they synchronize the clock offsets by means of a two-way message exchange.

4.5 Hardware in Wireless Sensor Networks

It is difficult to compare the performance of above mentioned protocols due to the varying parameters and depth of detail in the descriptions. Some of them claim to be superior in performance to others thanks to low level timestamping. However, would the same statement be valid if both protocols would use the same type of timestamping. Therefore, in the next sections the behavior of some protocols is investigated in detail when they would have been implemented on the same hardware and timestamping would occur at the exact same event. Since there tend to be differences in hardware and therefore also in sources of delay or variation, first the properties of three types of WSN hardware platforms are discussed.

4.5.1 Magnetic Induction Radio ASIC

The Magnetic Induction Radio ASIC (MIRA)[1] is a very low power, small footprint ASIC developed by NXP semiconductors, where a CoolFlux DSP (Digital Signal Processor) is collocated with a magnetic induction radio component. Only part of the functionality and possibilities of the ASIC is discussed here due to confidentiality agreements and the extensiveness of the ASIC. A reduced block diagram of the internals of the ASIC is depicted in Figure 4.7. The figure shows solely the relevant components, however, a wide range of additional peripherals are available. The program memory (PMEM) constitutes of 6 Kwords flash memory, while the X- and Y-memories represent the available 33 KB SRAM memory. The ASIC supports both an external clock source as well as an internal RC oscillator aided operation. The external crystal should have a value between 24.734 MHz and 38.144 MHz.

The radio part of MIRA is composed out of the radio front-end (the physical layer) and the low-level part of the MAC. The radio makes use of Continuous Phase Frequency Shift Keying (CPFSK) modulation, supports carrier frequencies from 7 MHz to 15 MHz and a maximum raw bandwidth of 298 Kbps can be attained. At reception, the captured magnetic field will induce a voltage across the external radio coil, which is tuned by means of an internal capacitor bank. The distance that can be covered depends on the transmission output power and the type of coil used as antenna, but ranges in the order of meters have been demonstrated.

The low-level part of the MAC, which is provided by the radio part of the ASIC, is inherently TDMA based. The radio comprises a 16 bit counter running at the TX bit clock frequency, which is called the superframe counter. A superframe defines the cycle period of the TDMA algorithm. By means of giving orders, which are in the form of writing register values, certain actions can be programmed at given instants of the superframe counter. The orders specify the type of action, the associated parameters and the time of activation. The type of action can consist of a receive or transmit command amongst others. The associated parameters can

[1]For more information contact by email nxh.info@nxp.com

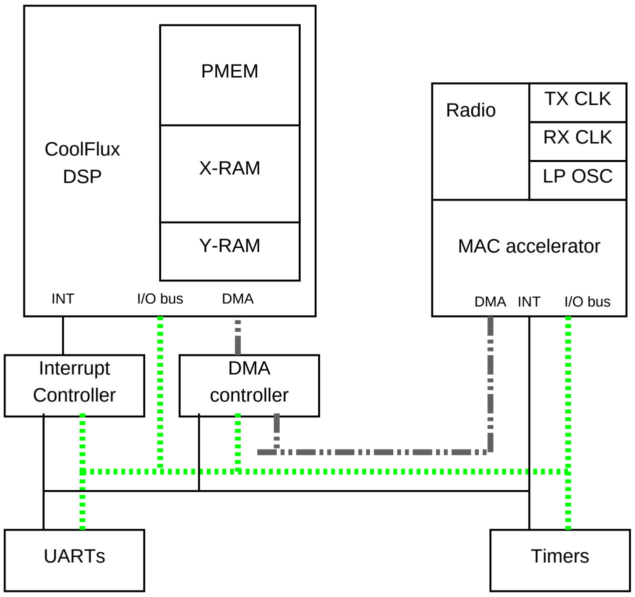


Figure 4.7: *Simplified block diagram of Magnetic Induction Radio IC(Ref: Adapted from Coolflux Transceiver Radio Specification datasheet, 2007 [113], Figure 1.)*

for example specify the DMA channel, and the time of activation is expressed as a value of the superframe counter. Since the superframe counter is running at the same clock rate as the bit transmission mechanism, the time of activation can easily be translated into the number of bits. The slots of the TDMA frame are determined by the scheduled orders. The evaluation of the orders is done when the value of the superframe counter equals the activation time of the order. Asynchronous orders are evaluated as soon as they are scheduled. Upon evaluation of a transmission order, the TX PLL and TX front-end become active, after which the actual packet is transmitted. Before transmitting the payload, a preamble and sync word is transmitted first, which are generated by the hardware. During the transmission of the preamble reaches the power on the antenna its maximum at the transmission side, while the receiver is able to detect the received signal level, perform gain control and adjust its bit sampling clock. Likewise, the RX PLL and Rx front-end become active upon evaluation of a reception order, after which the receiver performs adjustments to fine tune to the transmitter, thanks to the transmitted preamble. The following series of bits that is received consists of the frame-sync, which is compared to the expected frame-sync and the synchronization threshold. When a valid frame-sync is received, a snapshot of the superframe counter is taken and stored into a register for later use. After the reception of the sync word, the payload is received.

Neither the reception nor the transmission make use of buffering, the data is transferred directly to and from memory by means of DMA. Thanks to the DMA transfer, the DSP can unhindered continue operating and thanks to the prefetching of the first DMA word while the transmission front-end is being activated, the radio does not suffer from unpredictable data transfer times. Since the DSP is specifically designed to support audio algorithms, the word size of the memory is 24 bits.

The Radio part of the ASIC supports a considerable functionality, flexibility and configuration options, both low level as well as higher level frame management functions, that are not discussed here. Although this flexibility is extremely valuable, it is out of the scope of our research area.

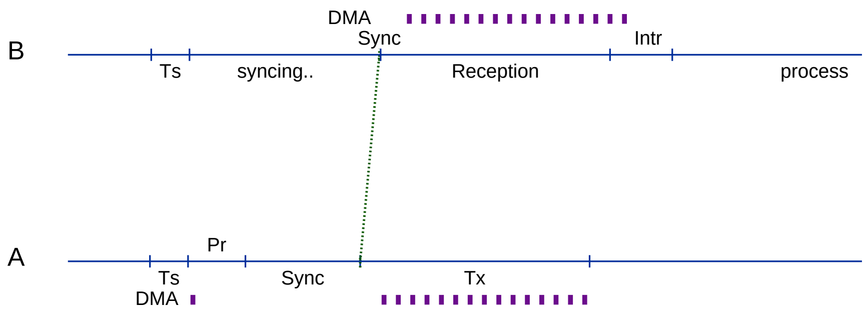


Figure 4.8: *MIRA Transmission and Reception timing*

Taking all functionality and timings into consideration of the MIRA, a timing diagram can be composed , such as Figure 4.8, which shows the possible sources of delay and more importantly, the possible variations on those delays. The timings are based on the transmission of 4 bytes data and an additional two bytes for the CRC. Note that the point of interest is with the different sources of variable duration, which also explains why the diagram does not show absolute time references, but relative time durations between the different events.

The time '*Ts*' indicates the required time to activate the radio, which includes amongst others starting the corresponding PLLs and activating the radio front-end. The Radio starts transmitting the preamble, indicated by '*Pr*' in the figure, after which it sends the sync word. As soon as the sync word is sent, the data can be transmitted. Note that the first DMA word has already been fetched while the radio was sending the preamble. Therefore, there is no delay on the data fetching and therefore also no variation, all data is available to be sent. All transmission timings are strictly defined by the register values, configured in the Radio and are configured on a per bit basis. Therefore, if there are variations on the execution of the transmission steps, they can be considered minimal.

The receiver does not have the luxury of knowing its timing as precise as the transmitter. After the activation of the radio, it needs to fine tune to the signals it receives and try to catch the sync word. There is a possible variation of several bits on the detection of the sync word, which signifies that a maximum deviation of ∼12 µs is possible.

4.5.2 TelosB sensor node

The TelosB sensor node platform is a popular platform in the Wireless Sensor Network research community. Unlike the Magnetic Induction Radio ASIC, is the control part and the radio part placed in two different ICs, without direct memory access, which requires them to communicate over an SPI bus.

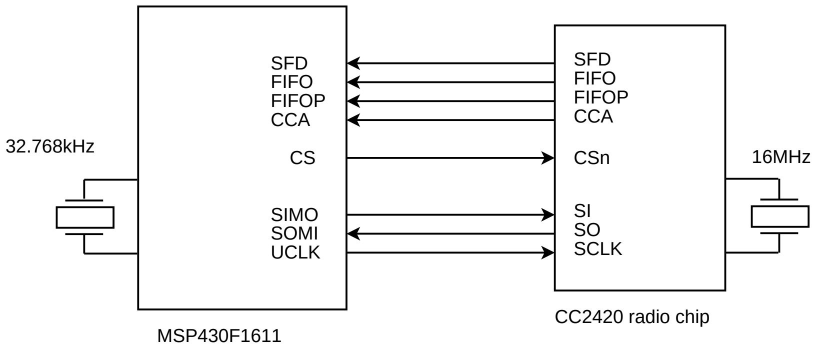


Figure 4.9: *Simplified block diagram of TelosB sensor platform. (Ref: Adapted from MSP430x1xx Family User's Guide, 2005 [114], Figure 4.1, page 115.)*

Figure 4.9 depicts the two major components and their interactions of a TelosB sensor node, where the MSP430 component is the microcontroller (MCU) and the CC2420 component is the radio chip. A crystal of 32.786 kHz provides a clock signal for the MCU, while a 16 MHz crystal provides the clock signal for the radio chip. The bus that is employed to transfer data between the two components is an SPI bus, which has the possibility to operate in a full-duplex bidirectional manner. Besides the SPI bus, there are also individual connections that can signal an interrupt, announce the state of a component or activate the component. The program that is loaded into the TelosB is executed on the MCU, which sends commands towards the radio chip to send data over the wireless medium. Upon reception of a packet, the radio chip signals an interrupt to the MCU, which takes the appropriate actions to transfer the received packet from the radio chip to its own RAM. In order to understand the interactions between the two components and detect possible sources of variability with regard to timing, it is imperative to understand the components individually. This discussion will elaborate on the two components and specifically on the relevant parts that could influence the timing of sending and/or receiving.

The MCU, MSP430F1611

The MSP430 [115][114] is a 16 bit RISC processor with a plethora of peripherals and a flexible clock system. The MCU is designed for ultra low power applications since it provides several low-power modes, where depending on the mode, a number of components are shut down. The clock system is a most critical component with regard to timing, which is the reason Figure 4.10 depicts the clock system in more detail. The figure shows a restricted view of the clock system, depicting only the configuration used in a specific implementation. The chip can operate with two external crystals, XT1, which is usually a low frequency crystal which can be controlled by the low power internal oscilator circuit, and XT2, usually a high frequency crystal. The design of the TelosB mote did not include a second crystal for XT2, which explains the lack of XT2 on the diagram. Besides the crystals the MSP430 also contains an internal Digitally Controlled Oscillator (DCO) which can act as a clock source.

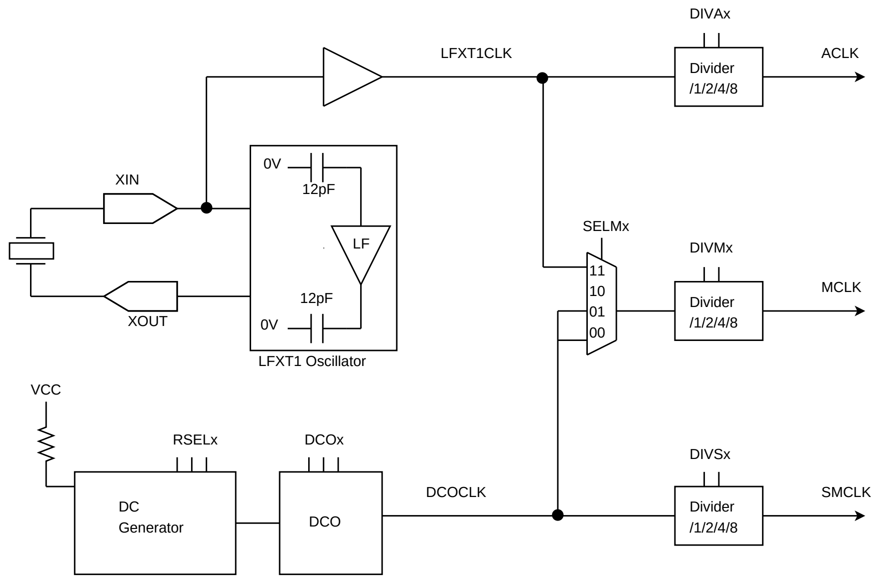


Figure 4.10: *Simplified clock diagram of MSP430F1611*

Three system clocks can be discerned in the figure, the master clock (MCLK), the sub-main clock (SMCLK) and the auxiliary clock (ACLK). The MCLK is used by the CPU and determines the length of CPU cycles. The SMCLK is used as an input for peripherals that require some sort of timing information. Due to the lack of XT2, this clock can only be sourced from the internal DCO clock. Each peripheral can determine a custom divider for itself to create the required clock speed. The ACLK can be used as a source for timers or just as a reference. The ACLK is a very stable clock signal, since it is derived from the external crystal. However, the resolution of this clock is rather low, 32.786 kHz, i.e. a period of 30.5 µs. This is low in comparison with the transmission rate of the radio, which is 256 Kbps, that is a period of 3.90625 µs per bit or a period of 31.25 µs per byte.

The specific setup discussed here selects the master clock from the DCO clock, which operates at its maximum speed, 4096 kHz. The SMCLK is also derived from the DCO clock, however, while the master clock uses no prescaler, the SMCLK uses a prescaler of 1/4, that is, the SMCLK is 1024 kHz. Of all peripherals, only the SPI bus influences one of the sources of synchronization error due to the transmission of messages [88]. The '*Send Time*' is influenced since the SPI bus, which operates at a maximum clock speed of half the SMCLK speed, is used to transfer packets from the MCU to the radio IC.

Like the Magnetic Induction Radio ASIC, the MSP430 provides DMA functionality. However, the MSP430 is required to shut down or at least reduce the CPU activity to 20% when allowing the DMA to transfer a block of data. As a consequence, incoming system interrupts remain pending and are processed only after the completion of the DMA transfer. Therefore, it is considered cautious not to use the DMA subsystem when interrupts are to be expected or the CPU is expected to perform some other tasks in parallel.

A last peripheral of the MSP430 that is discussed here, is the timer. The MSP430F1611 contains two timers, TimerA and TimerB. Those timers have an extensive arsenal of functionality, however, basically they are counters that increase their count every time a clock tick is given. The timer can be programmed to signal an interrupt at a specified time. Possible clock sources are the SMCLK and the ACLK. While the SMCLK is running at a higher speed and can therefore produce a higher resolution, the ACLK is more stable. It depends on the requirements of the application which of the clock sources is preferred. In the discussed setup, TimerB is sourced from the ACLK for its stability.

The radio component, CC2420

The CC2420 radio operates on its own crystal of 16 MHz and is able to transmit at 256 Kbps, that is, one byte every 31.25 µs. The radio makes use of a buffered transmit and receive mechanism, in both directions a 128 byte FIFO is available, sufficient for a single packet of maximum size. The associated MCU needs to send the to be transmitted data over the SPI bus towards a specific register, from where the radio chipset places the data at the correct place in the TX FIFO. Reading from the RX FIFO is similar, the MCU reads from a single register address over the SPI bus and each time the radio chip increments the read pointer internally. Specific commands, such as sending the transmit command, placing the radio in receive mode etc. can be achieved through the means of a single byte SPI transmission to a specific address. Upon receiving the command for transmitting the data, the radio activates the required PLLs and defines a fixed duration of 12 symbols, that is $\sim$192 µs, to start up the radio clocks. When the front-end of the radio is fully active, it first sends a preamble, which is of configurable length, usually 4 bytes, and a byte which allows the receiver to identify the start of a packet, the SFD byte. Upon transmitting or receiving the SFD byte, the line at the SFD pin changes its level, allowing the MCU to capture the edge transition as an interrupt trigger. This can be used to capture the current time of the timer located in the MCU.

TelosB timing

When combining the characteristics of both the MCU and the radio chip, the result is depicted in Figure 4.11, where the timing characteristics of a transmission and reception are shown.

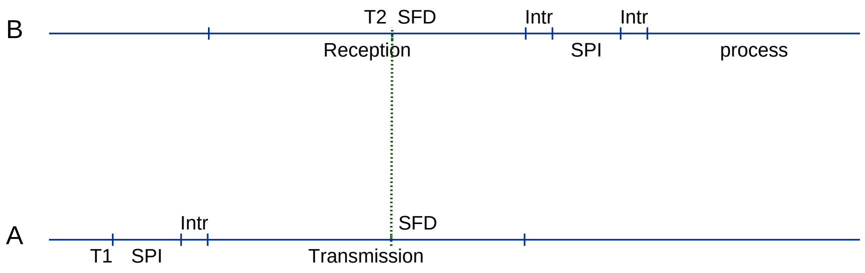


Figure 4.11: *TelosB Transmission and Reception timing*

Note that in this setup the timing parameters are taken as discussed in the previous sections and the SPI data transmission is performed entirely before sending the '*start transmission*' command to the radio chip. The timing diagram is based upon a transmission of eight bytes. When the SPI data transfer from the MCU to the radio chip has finished, an interrupt is generated in the MCU, which takes time to be served. Unfortunately, this time is variable and care should be taken to ensure timely interrupt handling. The moment the SFD is transmitted or received, a signal is generated at the transmitter or receiver respectively, allowing both nodes to record the actual time. According to the specifications of the CC2420 chip, the time lag between the two signals could deviate up to 2 µs. As soon as the receiver captured the complete packet that was sent, an interrupt is generated, which also poses some variability. The interrupt routine should trigger the function which will transfer the received data from the radio chip to the memory of the MCU, where the contents can be processed.

One of the major differences between the MIRA and TelosB platforms is that MIRA tranfers all data word per word during the reception directly to the right position in memory, where it can be directly accessed by the DSP, whereas the TelosB nodes need to first transfer the entire packet from the radio to the MCU, before any processing can take place. Moreover, the transmission of the MIRA is likewise arranged such that the radio can directly access the memory without interfering the DSP which can process the recently arrived packet. The TelosB node should first transfer the received packet from the radio chip to the memory of the MSP430, then transfer the to be transmitted packet to the radio chip, after which it can send the transmit command to the radio chip and finally can start processing the received packet. Note that the performance of the TelosB data transfer can be boosted by first loading a few bytes in the radio chip, then sending the send command to the radio and while the radio is starting its transmission, the remainder of the bytes are loaded in the radio chip.

4.5.3 CC2538

While there are up to date no sensor node platforms available with the next generation IEEE Std 802.15.4 compliant devices, such as the CC2538 chip[116][117], they provide interesting properties. Therefore, the differences of the CC2538 with the TelosB platform is shortly highlighted in this section, even though no timing analysis is performed with this hardware. The most innovative feature of the chip is that it provides a single chip solution, where the MCU is integrated with the radio chip into a single package. The advantage is that SPI transfers are no longer required, this can be achieved by means of memory transfers. However, a lot of additional adjustments are performed in order to optimize the performance. The CPU is more efficient and runs at higher clock speeds, while preserving the low power consumption. The radio section contains its own dedicated timer, which is running 1000 times faster than the transmission speed of a single byte. Upon reception or transmission of the SFD, a snapshot of this timer register is taken and stored into a dedicated register, available for later use. The DMA controller does not shut down the CPU like on the TelosB nodes and can be used to transfer bytes from memory to the TX or RX FIFO of the radio, thereby reducing this action to a hardware triggered action, where no interaction is required. Such changes in the architecture ensure a more efficient and more accurate operation.

4.6 Hardware related performance

This section discusses the influence of a possible implementation of the discussed protocols on a realistic hardware platform. From all protocols, three different methodologies can be distinguished, a receiver to receiver synchronization, a two way messages exchange synchronization and a receiver only synchronization. Figure 4.12 depicts a message exchange which comprises all types of communication with regard to the timing specifications of the hardware of the TelosB sensor platform. The figure shows a transmission of eight bytes and assumes the maximum SPI bus speed, that is, 512 Kbps.

The figure depicts all timings that were discussed in Section 4.5.2, where node A initiates the transmission at time T1, at which point the timestamp is taken and placed in the data. The timestamp mechanism is ambiguously defined in the related work. It is mentioned to take a timestamp in the MAC layer, as low as possible. Therefore, in this analysis, the timestamp is captured right before transmitting the packet contents to the TX FIFO of the radio over the SPI bus. The data transfer over the SPI bus from the MCU towards the radio chip is indicated in the figure as '*SPI*' and the following interrupt is indicated as '*Int*'.

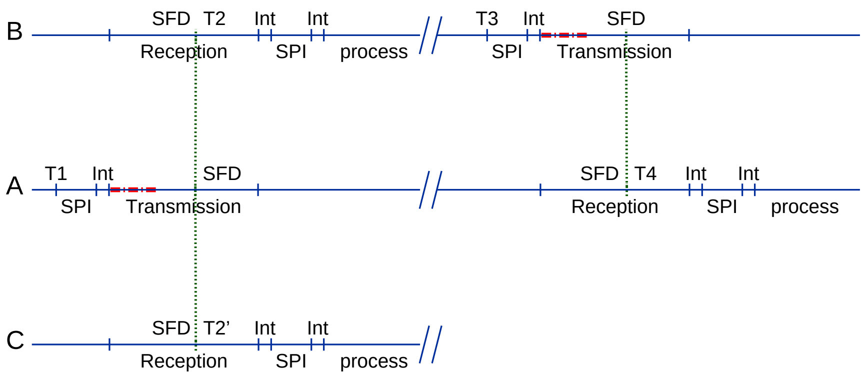


Figure 4.12: *2-way message exchange on hardware*

The interrupt handler triggers the start of the radio transmission directly by sending a strobe command to the radio chip over the SPI bus, or schedules a routine which performs this action. Upon receiving this command, the radio chip activates the related PLLs, upon which the radio chip FSM waits for 192 µs, which is marked as a thick dashed red line in the figure. When the PLLs are active, the radio starts the transmission of the preamble and sync byte, which is called the '*Start of Frame Delimiter*' (SFD). According to the specifications of the radio chip, there might be a time difference of 2 µs between the trigger of the SFD pin at transmission and reception side, which is also shown in the figure, however this time interval is far smaller than all other timings, such that it is hard to discern. Note that this time interval also comprises the propagation delay and delays from other sources, such as demodulation. Radio waves propagate at $3 \times 10^8 m/s$ in open air, which is equivalent to 30 meters per 100 ns, which is far less than the 2 µs the radio experiences when sampling the SFD pins.

At the reception side, the most convenient location for taking a timestamp is at the reception of the '*Start of Frame Delimiter*', SFD, which causes a rising edge on the SFD signal pin. When configuring the MSP430 appropriately, this signal can generate an interrupt, while capturing the current time. The trigger of the SFD reception allows both receivers B and C to take timestamp T2 and T2' respectively. After the reception of the data is completed, an interrupt is triggered, which introduces the transfer of the data from the RX FIFO to the MCU over the SPI bus. Upon completion of the SPI transfer, an interrupt needs to be handled, after which the processing of the data can be initiated. The transmission from node B to node A follows a similar pattern and therefore needs no further explanation.

The diagram shows that CSMA is not considered, since this would increase the uncertainty at the sender side. Some protocols take the variable access time into account by capturing the timestamp after the CCA routine has confirmed a free medium, however, this could lead to more complicated and time critical algorithms. Only receiver-to-receiver based synchronization algorithms would not experience any disadvantages. Possible improvements can be made to shorten the total transmission time by sending the transmission command even while not all data is yet loaded into the TX FIFO. While this shortens the total time, this does not eliminate the uncertainty intervals of the process.

The next subsections discuss the figure and respective timings for each type of synchronization methodology, that were discussed previously.

4.6.1 Receiver-to-receiver

Thanks to the elimination of the '*Send Time*' and '*Access Time*', the analysis of this type of protocol is rather straightforward. Based on the assumption the timestamp T2 and T2' is captured at the arrival of the SFD, it is known there is a worst case deviation of 2 µs between the two timestamps. This would imply that if all nodes exchange their timestamp, all nodes could become synchronized within those 2 µs and even less. The unfavorable characteristics of this type of synchronization are the tremendous number of messages that need to be exchanged and the lack of synchronization to the sender. This type of protocol works on a localized scale and becomes computation intensive on the scale of a complete sensor network.

4.6.2 Two-way message exchange

When taking into account the timings of Figure 4.12 and assuming $\varPhi$ is the immediate time offset between node A and B, then the times T2 and T4 can be defined by the following statements:

$$\begin{aligned} T2 &= T1 + t_{SPI} + t_{intr} + t_{preamble} + t_{SFD} + t_{SFDdiff} + t + \varPhi \\ &\quad and \\ T4 &= T3 + t_{SPI} + t_{intr} + t_{preamble} + t_{SFD} + t_{SFDdiff} + t' - \varPhi \end{aligned} \tag{4.9}$$

The variables can be defined as: t_{SPI} is the required to transfer the data from the MCU to the radio chi over the SPI bus, t_{intr} is the time required to capture and handle the interrupts, both from the SPI done interrupt signal as the SFD capture interrupt signal, $t_{preamble}$ and t_{SFD} are the times required to transmit the preamble and SFD bytes respectively, $t_{SFDdiff}$ is the possible time difference between the transmission of the SFD and the recognition of the signal, which is defined by the datasheet. Note that the definition of the latter in this formula does not include the transmission propagation, which is defined as t and t'. Based on the former statements, the offset of node A relative to node B can be formulated as follows:

$$\Phi = \frac{(T2 - T1) + (T3 - T4)}{2} + \frac{\Delta_{SPI} + \Delta_{intr} + \Delta_{TX} + \Delta_{SFDdiff} + t - t'}{2} \tag{4.10}$$

Unlike in the formulation of NTP [83][84], the deduction of the experienced delay by means of solely the timestamps is not that straightforward. The delay is dependent not only on the timestamps T1 till T4, but also on the interrupt handling times, the SPI transferal time, the transmit time of the preamble and SFD, the deviation between the transmission and reception of the SFD and the difference between the propagation times. Therefore, when employing a similar reasoning as NTP, the uncertainty factor, which is based on the RTT is not only dependent on the propagation delay, but also the duration of the SPI transfer, the interrupt handling duration and the duration of transmitting the preamble and SFD.

On each of those elements a deviation can be detected. Both Δ_{SPI} and Δ_{TX} are bound by their respective clocks and their relative frequency difference, which is in the order of 40 ppm for the radio and a somewhat lesser accuracy for the SPI clock source. However on such short time frame (156.25 µs SPI transfer and 343.75 µs preamble and SFD transmission), this deviation can be deemed to be negligible. Based on a clock with 40 ppm tolerance, the deviation of the radio transmission would be ∼27.5 ns, whereas the deviation of the SPI transfer, when assuming a tolerance of the resistor array of 1000 ppm, would be around ∼312.5 ns. In case the precision of the resistor array, and therefore the DCO clock, is assumed to be 10000 ppm, the deviation between SPI transfers could rise up to 3.125 µs. The major contributor to possible deviations is the interrupt handling time. When not correctly handled, the interrupt might be kept pending by other processes that have disabled interrupt processing for a short time.

A possible improvement could be achieved by capturing the timestamp T1 when the SFD pin signals the transmission of the SFD byte. When allocating some dummy bytes right after the SFD and length byte, there is sufficient time to transfer the 16 bit timestamp over the SPI bus to the TX FIFO before the radio starts transmitting these bytes. In this case, according to the specifications of the CC2420 radio, the difference between the two timestamps is maximally 2 µs, and therefore provides an excellent time reference. Moreover, this eliminates the deviations in SPI transfer speed and radio transmission speed. There is still an interrupt handling uncertainty, however, with careful planning of the system, such as reserving the interrupt system to serve only interrupts from the SFD pin at that specific moment, while preventing the disabling of interrupts, this could be reduced to an absolute minimum. The resulting offset equation would be formulated as follows:

$$\Phi = \frac{(T2 - T1) + (T3 - T4) + \Delta_{intr} + \Delta_{SFDdiff} + t - t'}{2} \tag{4.11}$$

4.6.3 Receiver-only

The receiver-only protocols that were discussed are based upon knowledge of the hardware. The timestamp is taken as late as possible, at moment T1, to ensure a minimal variance of in-between actions. The receiver is expected to derive the time delay between the capturing of the timestamp and the reception of the information by means of knowledge of the operation of the hardware. Based upon the information that time T2 is separated from the transmission of the SFD at most 2 µs, solely the delay between the transmission of the SFD and the capturing of T1 should be determined. Unfortunately, the SPI transfer and the interrupt handling determine the in-between variance and it is already determined in previous section that those two factors could influence the uncertainty a great deal. Nonetheless, this scheme is workable, certainly if the to be synchronized clock has a granularity of around 30.5 µs, as the external crystal of the MSP430 at the TelosB sensor node. Usually this crystal is taken as a time reference for its stability.

4.7 Slot based synchronization

This section describes a novel synchronization method, which reduces the required number of message exchanges to a bare minimum, while at the same time reducing as much as possible sources that could lead to a higher variance. The proposed methodology is based on the Receiver-Only methodology, however, instead of trying to synchronize on the captured and sent timestamp, synchronization is performed on the time of arrival of the sync word or SFD, depending on the naming conventions for the different hardware. From the hardware analysis in Section 4.5 can be deduced that the arrival of the sync word is very precise, in the case of the Magnetic Induction Radio ASIC (MIRA) only a few bits of deviation can be expected, while in the case of the TelosB platform the deviation is even reduced to a few microseconds. With accurate knowledge regarding the transmission timings, the time to trigger an action can be predetermined, such that the sync word arrives at the specified time.

Note that this type of protocol does not attempt to provide external synchronization, that is, synchronization relative to a standard clock format. The only type of synchronization performed here is time synchronization, where an attempt is made to synchronize the local timers of the sensor nodes, such that medium access methods such as TDMA are made possible. Some of the related work seems to assume a CSMA based medium access, however, if accurate time synchronization is mandatory, a controlled environment is required and therefore, a TDMA scheme.

The symbiosis between the TDMA scheme and the protocol becomes apparent when describing the message communication. The sender transmits at regular intervals, for example once every superframe, a sync packet, containing the slot relative to the superframe in which it will be sent. This type of information can be scheduled, is known beforehand and therefore requires no time critical actions. A timer is scheduled to trigger the start of the transmission at the predefined time, such that the sync word arrives at the required time instant. The receiver is able to deduce the time of the sender by means of the slot information embedded into the packet payload. Therefore, without the TDMA scheme, the synchronization would not be possible and vice versa.

Figure 4.13 depicts the process of the synchronization, where node A is sending a synchronization message in the same slot each TDMA frame. Node B, which is attempting to synchronize to node A, listens for sync packets for a certain time. Upon reception of such packet, the expected arrival time of the packet is calculated. Based on the information of the expected arrival time and the actual arrival time, the new frame boundary can be deduced. Note that the synchronization is a fast and trivial process.

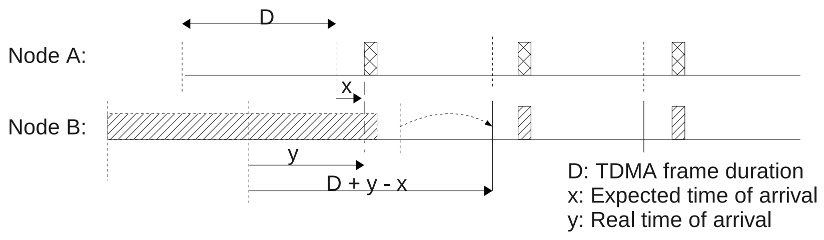


Figure 4.13: *Synchronization process*

In order to evaluate the performance of this protocol, it has been implemented on two types of hardware, the MIRA and the TelosB platform. The implementation details of the synchronization protocol are discussed in the next subsections, where the possible sources of uncertainty are considered.

4.7.1 Magnetic Induction Radio ASIC

As previously mentioned, the MIRA provides a superframe counter that operates on a bit level, that is, its finest granularity is equal to the time required to transmit a single bit. Orders for transmission or reception can be placed and scheduled at specific times, based upon this superframe counter. The action of comparing the order activation time with the current value of the superframe counter is a background process which is being handled by the hardware. Therefore, no interrupt handling uncertainties are generated by this process. Since the duration of the several elements within the entire transmission chain is expressed in number of bits, the estimation of the time at which the sync word is sent can be precisely predicted. Based upon this information, the activation of the order can be scheduled such that the sync word is sent at a specific time. Since the structure of the TDMA superframe is known to the whole sensor network, the transmission time of the sync word can be easily translated into the slot number, which can be regarded as an abstract representation of the time.

In order to deal with any uncertainties of the transmission or sync word detection, the receiver hardware defines a synchronization window, during which the sync word is to be expected. The window is typically in the order of 12 bits. When no sync word is detected during this window, a predefined action is undertaken, which means usually that the reception is aborted. Upon detection of the sync word, a copy of the current value of the superframe counter is stored in a designated register by the hardware. No user interaction, nor interrupt handling is required, thereby eliminating any possible uncertainties in the operation. The payload of the sync packet contains the slot number in which it was sent, thanks to which the receiver can correlate the expected arrival time of the sync word to the recorded arrival time. Based upon this information it is a straightforward operation to adjust the frame length for a single cycle, such that the super-frames of both nodes start at the same time. As can be noticed, the synchronization methodology ensures a synchronization with a precision of the size of the synchronization window, which is 12 bits. Note that this methodology is suitable for an initial synchronization only. The adjustment of the superframe can result in a drastic change in superframe size, thereby hampering the regular data transmissions. Consecutive synchronization attempts should only adjust the superframe counter with one tick forward or backward, thereby not disturbing the regular operation of the system. When the difference between expected time and calculated time is far larger than could have been expected, an adjustment, such as the one used during the initial synchronization, is made based on the obtained timings, thereby resynchronizing the two nodes.

4.7.2 TelosB

Unlike MIRA, the processor element and the radio component are separated on the TelosB platform. Each component handles its own actions and there is a need to exchange data between the two components. Moreover, the radio component of the TelosB platform does not have an embedded counter that is in sync with the bit transmission, i.e. there is no superframe counter, at least not one that is accessible to the MCU. As a result, the MCU needs to provide for its own time reference in order to determine the slot boundaries of the TDMA schedule. However, the MCU execution speed is determined by a different clock source. The processing of instructions by the MCU is sourced by the DCO clock. Likewise, the peripherals, such as the SPI bus, are also clocked to the DCO clock, which is an RC oscillator and provides therefore a low time accuracy. When employing this type of clock as a superframe counter, the worst case situation would result in a time offset between two nodes of 2 ms every second, assuming the precision of the resistor array equals 1000 ppm. In terms of a radio transmission, 2 ms is equal to a difference of 64 bytes. Such time offset between the two nodes is unacceptable and would require both nodes to resynchronize every 15 ms in order to maintain the synchronization within a single byte difference. Such operation is not feasible if the network should serve some other functionality besides synchronization, which is usually the goal. In order to obtain a reasonable synchronization interval, a high precision clock should be used. Unfortunately, the TelosB platform provides only a single clock which has a sufficient tolerance, however it is a low frequency crystal. The oscillator is able to generate a single clock tick every 30.5 µs, which is almost the time to transmit a single byte. Despite the mismatch in the frequency and the low resolution, this clock source is the most appropriate source to trigger the step of the superframe counter. Although the mismatch in frequency is in the order of 2.5%, it does not hinder the operation, since the two clocks operate in an isolated manner, provided that the slot sizes take into account the actual transmission time.

The transmission path of the radio component is strictly defined, allowing to predict very accurately the duration of time from the TXON strobe command until the '*Start of Frame Delimiter*' (SFD) is sent. Thanks to the abstraction of time in the form of slot indication, the sync packet can already be preloaded into the TX FIFO, while waiting for the precise time to start the transmission sequence, i.e., no timestamp information needs to be embedded in the packet payload. This is depicted in more detail in Figure 4.14, which indicates that no strict timing is required between the transfer of the data over the SPI bus and the start of the transmission. Unfortunately, due to the timer functionality of the MCU, which controls the superframe counter, a hardware interrupt needs to be serviced upon the elapse of the radio activation time, causing an extra source of uncertainty at the sender side.

As the hardware provides a certain synchronization window for the MIRA, so does the software for the TelosB platform. The receiver is activated such that there is a window during which the SFD is to be expected. This uncertainty window is the size of a few bytes. When no SFD is detected within this window, the reception is aborted. Upon the reception of the SFD, the signal on the SFD pin goes high, which triggers an interrupt at the MCU. The interrupt handler results again in some uncertain time duration while taking a snapshot of the current value of the superframe counter. Like with the MIRA, the expected arrival time is deduced and compared to the actual arrival time, which leads to the adjustment of the superframe counter.

Due to the construction of the TelosB sensor node, more sources of uncertainty are available compared to the MIRA platform. However, compared to the operation of related works on the TelosB platform, the possible variance introduced by the

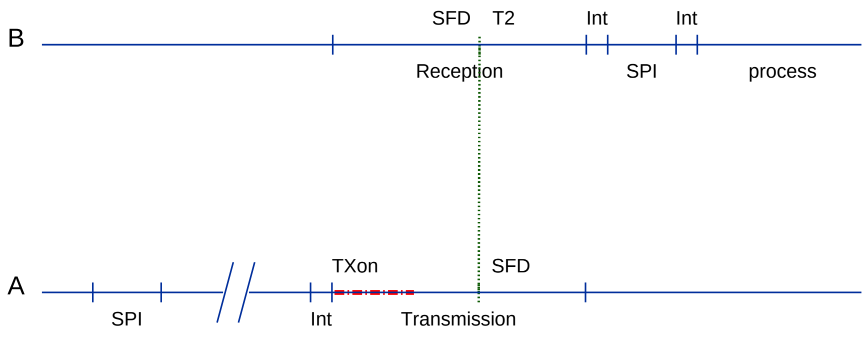


Figure 4.14: *Slot based synchronization on hardware*

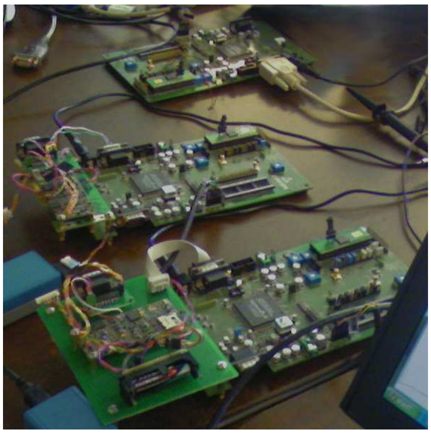
capturing of the timestamp and the start of the SPI transfer and of the SPI transfer itself is eliminated.

4.8 Proof of Concept

As a form of evaluation, this section describes the completion of a Proof of Concept (PoC) where a TDMA based protocol is implemented on the Magnetic Induction Radio ASIC (MIRA), with the addition of the proposed synchronization mechanism (Section 4.7). The synchronization protocol is also used to aid an alternative TDMA based protocol on the TelosB hardware, which is described in detail in Chapter 5.

The MIRA is especially suited for Body Area Networks (BANs) because of its small communication distances, thereby limiting the exposure of confidential information. Moreover, thanks to the magnetic radio, the wireless communication ensures a low radiation. Instead of penetrating the body, the magnetic radio waves rather follow the curves of the body. The PoC was then also designed in the context of a health monitoring application. The goal of the PoC was to capture and transmit ECG and accelerometer data over a multihop sensor network toward a sink, which would forward the data to a secure server. A snapshot of the system in operation is shown in Figure 4.15, where at the left side of the figure the three development boards are depicted, while the right image shows the received ECG signals.

The implemented TDMA protocol is derived from the Cicada protocol [32]. The protocol is developed specifically for Wireless Body Area Networks (WBANs), where a single sink is utilized. The routing section of the protocol is reduced to following the branches of the constructed spanning tree. The protocol employs a dynamic frame length, where all nodes are assigned a single slot downward in which the


(a) *hardware setup*

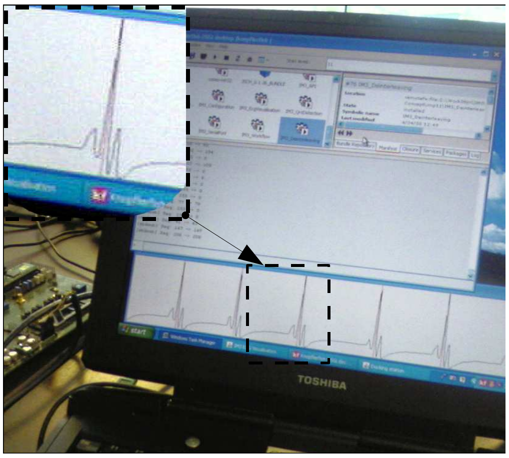

(b) *ECG output visualization*

Figure 4.15: *PoC setup*

parent indicates to its children the slots assigned to each of them. During the data sub-cycle, the nodes use the slots assigned to them to forward data and request bandwidth in the next cycle. In such way, the protocol achieves a traversal of the complete tree in a single cycle. A contention slot is made available for new nodes to be able to register themselves.

Due to the assumptions and methods employed by the Cicada protocol, its implementation is not straightforward. Some design choices cannot be used in a reliable manner, such as the dynamic frame length, due to for example a variable number of slots, which is an unreliable method and should not be applied. Therefore, a derived protocol is designed and used as a reference for the implementation. This section does not handle the specifics of the TDMA protocol, since the description would be out of the scope of this section. Instead, a high level description is provided of the association and data transmission section, since the focus of this discussion is on the synchronization section. As already can be noticed, the TDMA protocol consists of three phases: a synchronization, association and data phase. Those phases define the different states in which the sensor node is situated. Each of the phases is assigned a number of associated slots. The network could operate in the data phase, while a new node is still in the sync phase, trying to synchronize to the network. To avert collisions or interference, the slots associated to the different phases are not shared between the phases. The logical steps in the state of a sensor node are synchronization, association and data exchange. During the synchronization phase a node attempts to detect other nodes that already participate in the network and synchronize to them. The association phase depicts the state in which the sensor requests a slot from its selected parent node, that is, the node to which is has been synchronized to. The data exchange phase indicates the node has received its personal data slot and can start sending sensor data in its designated slot.

The synchronization phase has been assigned four synchronization slots at the start of the superframe. A node operating in the data phase employs two sync slots, one to listen to incoming synchronization messages, the other to send its own synchronization message. In order to reduce the required number of slots for the whole network, spatial reuse of the slots was opted. Otherwise, the number of sync slots would equal the number of nodes, which is not necessarily a fixed number, especially when considering the scalability of the network. The network is allowed to have dynamic allocation, however a dynamic frame length, due to for example a variable number of sync slots, is an unreliable method and should not be applied. The chosen method is a flexible solution which does not limit the number of nodes in the network and yet has a relative small overhead compared to assigning a slot to each node. Because of the usage of multiple synchronization slots, the payload of the sync packet should indicate in which sync slot the packet should have arrived to deduce the time offset relative to the sender.

During the synchronization phase, nodes listen to the medium and collect information regarding received synchronization messages, such as slot information, frame offset, signal strength and number of lost syncs. Based upon this information, the node is able to select the most optimal node as its parent, adjust its own superframe boundary to that of its parent and send it an association request. As a response to the association request, the node receives a message containing information in which data slot it is allowed to send its data to its parent. Upon reception of this information, the node enters the data exchange phase, where it will first select an unused synchronization slot to broadcast its own synchronization messages. If no free synchronization slots are available, the node stays quiet during the synchronization slots, since there are plenty of other nodes in the neighborhood that can fulfill the role of parent for future nodes that wish to join the network.

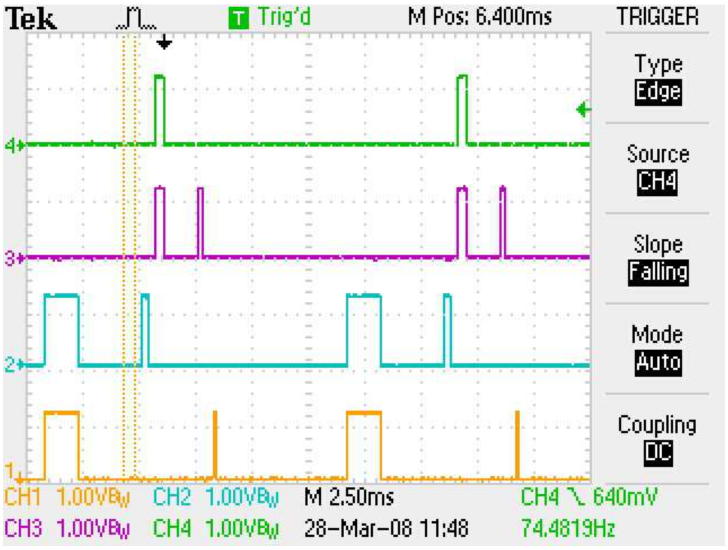


Figure 4.16: *PoC scope communication detail*

As a rule, all nodes keep listening to the synchronization slot in which their parent sends sync packets, not only to keep synchronized, but also to be informed if the parent is out of range. If such event should occur, the node returns to the synchronization state and tries to resynchronize to the network.

The operation of this protocol and its synchronization is depicted in Figure 4.16, which represents a snapshot of an oscilloscope, attached to the development boards. The pins that were measured are hardware driven and output a high signal when the radio is actively transmitting or receiving. The figure shows the communication between the root node and its child, where probe 4 and probe 1 are attached to the TX and RX pin of the root node respectively. Probe 2 and probe 3 are connected to the TX and RX pin of the sensor node respectively. The black arrow, at the top of figure, which indicates where the trigger has occurred, points to the transmission and reception of the sync packet of the root node and child node respectively. The next small peak that was captured on probe 3 indicates an attempt to receive an association packet, however, no packet was received at that time. This is a normal occurrence since nodes do not keep on sending association packets. The wide block captured by probe 2 indicates the transmission of sensor data from the child to the root node. Although the resolution of the snapshot is relative low, it is noticeable that the transmission and reception of the packets are aligned nicely.

In order to verify the entire TDMA protocol, the MIRA requires some ECG data input and accelerometer input. The PoC setup makes therefore use of rather large development boards, depicted in Figure 4.17, which not only contain the small MIRA board, but also a plethora of control and debug interfaces, such as SPI, I2C, UART and debugging tools in order to aid the development of protocols. The MIRA PCB is comprised of the MIRA chip, a power regulator, an external magnetic coil and several interfaces. A sensor board is attached to the development board by means of the UART interface. The default state of the sensor board is in idle state. As soon as the start command is sent over the UART, the sensor board starts sending ECG and accelerometer data at regular intervals, such that the MIRA receives at least 250 samples per second (500 bytes per second) ECG data.

The PoC (Proof of Concept) demonstrates the synchronization, loss of connection and resynchronization by means of the scenario depicted in Figure 4.18. In the scenario three different nodes are depicted. The root node, which is not equipped with any sensors and two sensor nodes, of which one is equipped with an ECG sensor, while the other node has an on-board accelerometer sensor. Because of the lack of sensors at the root node, its sole function is to receive sensor data from its children and transmit it towards the medical hub. The medical hub, represented by a regular laptop in the PoC, is designed to forward the sensor data to a remote server to store and process the data. In order to demonstrate the data gathering process, the PoC laptop which acts as the medical hub also processes the received sensor data and provides a visualization of the sensor data. Initially, the root node is directly connected to the node with the ECG sensor as depicted at the top of Figure 4.18. By moving both nodes farther from each other, the link becomes weaker, CRC

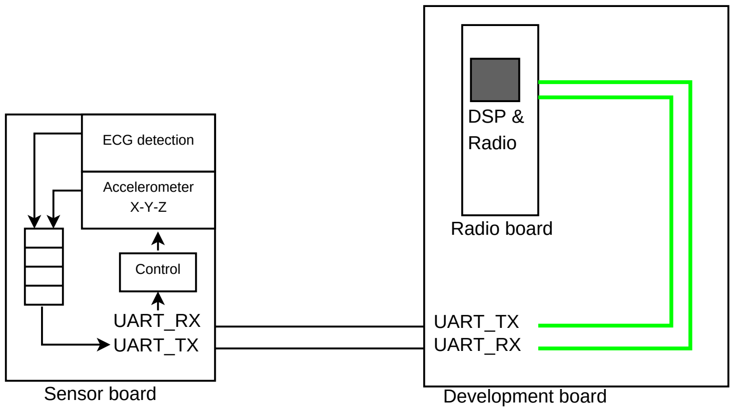


Figure 4.17: *Simplified block diagram of Magnetic Induction Radio IC platform*

errors start to happen, the child will miss synchronization messages and eventually the link is completely broken. This state is depicted in the middle of the figure. When a third node is inserted between the two original nodes, it first associates to the root node and then starts its own broadcast of sync packets. Those sync packets act as a means to synchronize to the network again from the point of view of the node with the ECG sensor attached. From the moment the last node is associated with the middle node, the middle node will act as a relay to forward the ECG data coming from the last node. The stability of the system was verified for a duration of several hours, thereby proving the accuracy of the synchronization protocol.

4.9 Conclusion

Time synchronization is often required in Wireless Sensor Networks. Not only scheduled transmission and reception schemes require a precise time reference, but the application might also need to be able to correlate the measurements of the different sensor nodes. This chapter elaborates on the sources that cause the phenomenon known as clock drift, which can be considered to be composed of two components, the clock offset and the clock skew. The clock skew can be considered as the deviation from the perfect frequency. Since hardware components are constructed in a real production environment, where a perfect production process is non-existent, every hardware component exhibits some deviation from its specifications. It is said that hardware operates within a certain tolerance to its specifications. Because of the specifications, it is clear that an oscillator which is based on a quartz crystal has a far better tolerance compared to an RC based oscillator.

Synchronization is an area which has attracted a multitude of researchers since many decades. The most interesting and relevant works have been described, since although those works were not designed for Wireless Sensor Networks, they still can

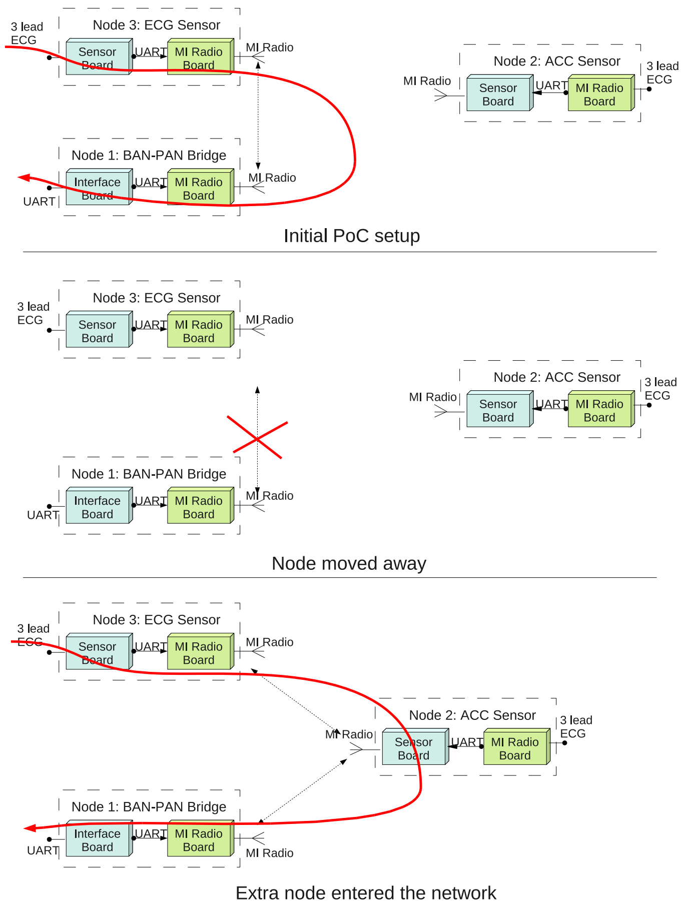


Figure 4.18: *PoC setup: extra node entered*

provide valuable insights in time synchronization in itself. A subset of the discussed synchronization protocols for WSNs in fact make use of one of these works. The works specific for WSNs can be cataloged according to the role of the sender and receiver within the synchronization protocol. As such, there is a receiver-receiver, a sender-receiver, and a receiver-only based synchronization. The protocols in the first class allow the receiver nodes of the synchronization message to become synchronized to each other, without synchronizing to the sender. The second class proposes methods for synchronizing the sender with the receiver by means of a message exchange mechanism. The third class of protocols synchronize the receiver to the sender by means of messages coming only from the sender. The last class requires knowledge about the specifics of the hardware.

As the different protocols use different criteria to define their superiority towards other protocols, it is hard to make a direct comparison. Therefore, this chapter discusses all three categories in the context of two sensor node hardware platforms.

The discussion shows that a great deal of the synchronization protocols exhibit a certain dependency on the hardware platform, except for the first class of protocols, which are designed to reduce the number of dependencies. Their downside is the lack of parent-child synchronization, which is imperative for scheduled transmissions and receptions. The multitude of the protocols within the second class makes use of the two-way message exchange, which shows a promising performance when being considered in theory. However, in practice it is clear that certain factors were not taken into account and could result in a deterioration of the precision. The last class of protocols consider knowledge about the specific hardware in order to ensure a minimal influence of sources of variable delay.

The proposed synchronization algorithm eliminates the dependency on the timestamp gathering time as in most discussed related protocols. Instead, the time is expressed as the slot index number, which is known by all nodes in the network, thanks to the fixed frame structure. The accuracy of the protocol mostly depends on the transmission time accuracy. In order to show the advantages of the protocol, its operation is discussed for both mentioned hardware platforms and is implemented as a Proof of Concept on one of the platforms in this chapter and is used as synchronization protocol to develop a different protocol in the next chapter.

CHAPTER 5

Fair Scheduling MAC in Wireless Sensor Networks

5.1 Introduction

As already discussed in Chapter 3, sensor devices that are able to operate in a Wireless Sensor Network (WSN) are prone to be small, low power and low cost devices. Therefore, these devices tend to have limited resources. Such resource limitation is reflected in a low amount of working memory, low amount of program memory, low processing power, low power MCU, etc. Algorithms running within a WSN need to utilize these resources as efficiently as possible. This Chapter describes the contribution of two different protocols and the analysis thereof in terms of latency and buffer size, starting from Section 5.4

One such resource is the remaining energy, which should be preserved as much as possible by applying power control on both the MCU and the radio component. When no tasks need to be performed for a long time by the MCU, it is common to allow the MCU to enter a low power state, where the power consumption is reduced to an absolute minimum. The radio component in its active state, i.e., either in receive or transmit state, is in most cases a large consumer of energy. As a derivation later in this chapter shows, is the energy required to transmit one byte, more than 170 times larger than the average energy required to execute one instruction. Therefore, when no data needs to be transmitted or received, the radio is placed in a low power state, where the power consumption is reduced. Chapter 3 already discussed a considerable number of methods, both contention and schedule based, that allow for energy preservation. However, while the discussed contention based protocols make an attempt to reduce idle listening as much as possible by introducing duty cycles and Low Power Listening, a complete elimination of idle listening is not feasible by means of these methods due to the lack of an accurate transmission schedule.

An alternative method is Time Division Multiple Access (TDMA), where a precisely controlled slot access is provided. Such scheme maximizes the power efficiency by placing the radio and MCU in low power mode when no action is required. On the other hand, the allocation of such schedule requires extra control overhead and time synchronization in order to align the slot and frame boundaries. A fixed frame duration, and therefore also a fixed number of slots is for practical reasons more appropriate as shown in the discussion in Section 3.2. However, this makes it harder for the TDMA protocol to anticipate a certain dynamic behavior of the network, such as the joining or leaving of nodes to or from the network. Moreover, for a homogeneous network, where all sensors need to send data at the same rate, a fair schedule could be realized by means of a trivial Round Robin scheduling scheme. The network can be designed such that the slot sizes match the precise data rate requirements. However, the application of a fixed slot size to a network composed of heterogeneous sensor nodes raises new issues. A slot size optimized for a specific data rate will result in either a tremendous packet fragmentation or a waste of bandwidth due to the reservation of a large time slot for just a small amount of data. Note that the fragmentation of packets should be limited, since every packet transmission amounts to extra overhead in terms of physical and MAC header information, while the amount of data per transmission is limited. The Round Robin scheduling would ensure a fair bandwidth allocation between the different nodes when using fixed slot sizes, that is, every node is assigned the same bandwidth. However, this would introduce large latencies for the high throughput devices. Likewise, the waste of bandwidth by the oversized slots for small data transmission also results in the usurping of the wireless medium, preventing more urgent or high throughput transmissions to seize the medium. As an alternative, other scheduling algorithms that are discussed in Section 5.3, such as Weighted Round Robin [118] or Weighted Fair Queuing [119] could be used, but each of them presents its own challenges for them to be used in a TDMA access mode with heterogeneous sensor nodes. Moreover, those works were designed in order to solve the contention problem that existed within the router of wired networks and were to interact with the higher layers of the network stack. Some protocols offer a guaranteed bandwidth, but it could vary, depending on the number of streams. The protocol that is being proposed in this chapter, is a MAC protocol, which is ensuring a collision free access to the wireless medium. Moreover, the proposed protocol is supposed to work in a stable manner, that is, once a certain scheme has been decided, it does not need to change anymore, even when other nodes enter or leave the network. On top of that, is the bandwidth allocation independent of other nodes, provided there is sufficient capacity within the network.

The challenges that need to be resolved during the design of a TDMA protocol therefore include the fixed frame structure, while at the same time a flexible number of nodes and variable transmission rate should be provided. The synchronization ensures an additional overhead, although it is often required even in contention based system to ensure some correlation between the collected data packets. Moreover, the precise scheduling of the transmissions and receptions makes a TDMA protocol more complex and often requires some extra control communication overhead. Chapter 3 discussed several slot allocation methods for scheduled transmissions. However, most of those methods consider only a single slot per node per frame, thereby renouncing any consideration regarding different data throughput requirements.

Irrespective of all technical difficulties and constraints, most applications that are aided by the data collection of sensor networks require to have the most recent data at their disposal. Providing a protocol that ensures fair scheduling on a heterogeneous sensor network under the previously discussed conditions can be a daunting task. Inspired by the research results of the protocol that was developed as part of the PoC which was discussed in Section 4.8, this chapter presents two types of protocols that take into account a heterogeneous transmission pattern of the sensor nodes, while ensuring a fair scheduling of the data. The design of both protocols takes into account practical constraints, such as memory consumption, processor speed, communication overhead, etc.

The protocols are specifically designed to operate in a tree, star or cluster based topology, where all nodes are constant bit rate sources, that is, there is no processing of the data on the sensor, or when there is, the outcome is a bitstream with constant bit rate. Both design decisions are discussed in more detail in Section 5.2. Both protocols make use of a centralized slot organization mechanism, located at the parent node or cluster head in case of a clustered network. The network is expected to be of limited size where the majority of nodes can hear each other. Therefore, the discussed methods do not consider spatial slot reuse in the strict sense of its definition, that is, all nodes are assigned a unique set of slots in order to prevent collisions. Note that the protocols support the sharing of slots between siblings, which is meticulously controlled by the parent node.

The protocols ensure a fair scheduling of the data, based upon the requested data rate. Based on this estimation of the required bandwidth, the root node determines a slot allocation which takes into account the repetition of the pattern, the heterogeneous character of the network and prevents the obstruction of sensors to gain access to the medium by a single sensor node with a large throughput. The distribution of the available bandwidth over the requesting nodes by the protocols is based on the application of the operational principle of Pulse Width Modulation (PWM) to WSNs. Instead of assigning the requested amount of bandwidth, the node is assigned an integer number of slots for only a fraction of the time. The utilization time fraction is determined such that the average bandwidth is equal or nearly equal to the requested bandwidth. A parent node distributes its available bandwidth between its children by allowing them to send in the same data slot, but in different TDMA frames. Moreover, in order to prevent the seizure of the wireless medium by a high throughput transmission, the slot allocations are dispersed. Granting half the available bandwidth to a node could be achieved by assigning a set of sequential slots, equal to half the number of slots per frame, to the requesting node, or it can be assigned every second slot. The latter results in an evenly distributed allocation, which allows other sensors to access the resource at regular intervals. The allocation is therefore more fair. Note that this method also provides a better response time.

The dynamic behavior of the network is considered by devising a schedule update method, composed out of resource allocations that consist of repetitive sequences, such that the existing transmission schedule is not affected by the extra resource assignment. Such approach and the need to receive the update, which is specifically for a certain node, only once also takes into account the lossy medium that poses no guarantees whatsoever on the arrival of the data. The schedule does not need to be repeated every cycle and if an update message is not received correctly, this does not affect the remainder of the network, or even the remainder of the schedule, just the newly requested resource allocation. An interesting property of this methodology is a perfectly predictable latency, which means that the protocol is suited for real-time applications.

The following section provides a discussion of the specific design decisions to which the proposed MAC protocol should comply to. The research that is related to the discussed topics in this chapter is presented in Section 5.3. Section 5.4 discusses the first methodology, which makes use of a straightforward slot allocation method based on the greatest common divisor (gcd). Although the employed principle is trivial, it exhibits some disadvantages, which are alleviated by the second protocol, of which a theoretical analysis is discussed in Section 5.5. The slot allocation method of the second protocol is more sophisticated and exhibits some promising properties, such as a deterministic latency. Therefore, the remainder of the chapter discusses the performance of this second protocol, first by means of a theoretical analysis, where the arrival of new data is represented by a linear function. A second analysis, Section 5.7, considers a more realistic data arrival in bursts, while a third analysis considers the performance and impact of the protocol in a networked environment. All previous analyses were performed by means of simulating the slot scheduling, which involved certain assumptions which need to be adjusted to more realistic terms in a protocol implementation. Therefore, Section 5.9 describes a complete TDMA protocol based on the scheduling protocol. The last section concludes this chapter.

5.2 Design Decisions

In Wireless Sensor Networks, MAC protocols van be classified according to their topology: tree based protocols and distributed protocols. The tree based protocols assume the existence of a root node, usually the sink, or one of the sinks, which starts building a tree network topology based upon a parent-child relationship. The distributed approach is based upon the idea that there is no central coordination possible in WSNs to construct a topology, although there always needs to be some kind of sink node, but might not be available at all times. Therefore, all sensors are deemed to act in a distributed manner. Note that a star based topology is a special case of a tree based topology where only a single level is available. The organization within each cluster between the members and the cluster head can possibly also be considered as a tree or star based topology.

Both types of protocols have their pro's and con's. The tree based topology requires time to construct, needs to rebuild a branch of the tree upon failure of a single node and has an obvious bottleneck, although this is a characteristic of WSNs in general, since the data usually flows to a single sink. The advantages of a tree topology are its intrinsically clear construction, where the routing path is usually determined by the branches towards the root node. This allows for a trivial routing method, a predictable performance and therefore also accommodates for its optimization. The distributed approach provides no clear topology, nodes start creating their own subnetworks within the network and eventually those subnetworks merge into one complete network. A perfect example of this approach is S-MAC [22]. Such network is more resilient to node failures, since there is no predefined path which needs to be followed and usually the network intercommunication is setup more rapidly. The disadvantage is the uncontrolled evolution of the network topology. Moreover, a routing protocol will need to define the path towards the sink node, instead of simply following the branches towards the root.

This chapter focuses on the tree based topology because of all the positive arguments provided above. Moreover, while a distributed sensor network is interesting from an academic point of view, a commercial environment requires a stable and reliable sensor network. Actual real-life sensor deployments for research purposes, such as [70][120][121][122][123], also point in this direction, where a rather simplified topology is applied. Often these networks are set up in a star formation, a fixed mesh network or employ a static routing scheme. Real-life deployments do not require very complex algorithms, they require stability and reliability, which can be easily be compromised when the algorithm becomes too complex.

For small scale networks, a tree topology type of a network can provide such properties, that is, a stable and reliable network, while still being sufficiently flexible to allow for the removal or the arrival of nodes. Larger scale networks are more likely to be divided into local clusters, where each cluster is operating in a tree topology of one or at the most a few tiers, while the cluster heads themselves can either be connected directly towards the sink, or they can be configured as a mesh network. Such network is also more interesting with regard to the energy usage and therefore battery lifetime of the sensors. The sensors in single clusters are expected to have the same battery lifetime, while the cluster heads have a different lifetime, except when a rotating cluster head selection algorithm is used to distribute the power usage, such as in [56]. When the application allows it, for example when the connected device itself is line-powered, the cluster heads are not battery-powered at all. An excellent example is presented in [124], which discusses the commercial applications of Wireless Sensor Networks using ZigBee. Wireless Sensor Networks operating in a

Home Automation environment are preferred to have the distinction between sensor nodes and forwarding nodes. Such distinction is more clear to the end-user, but also enhances the reliability and predictability. The reliability is enhanced since a faulty sensor node does not necessarily break the network connection. The forwarding nodes, i.e., the cluster heads, provide the network connectivity towards the sink. The predictability is improved since the sensors have a similar lifetime when all operate at the same duty cycle.

Another distinction that can be observed between WSNs is the data processing location. Some setups prefer to perform a preprocessing on the sensor and sends only the related information towards the sink. Other setups send all captured information as raw data towards the sink. Both methods have their advantages and disadvantages. The advantage of processing the data at the sensors is that less data needs to be transmitted and therefore less energy is expended due to the transmission of data. On the other hand, typical sensor nodes consist of an MCU with limited resources. Therefore, hardware support for complex computations is beyond the simple instructions set of such MCUs, software sequences need to complete the complex computations. Such complex computations can require a considerable number of instructions and thereby also consume energy. The transmission of the raw data is computationally wise trivial, however, requires a continuous transmission of the data.

In order to have an idea of the relation between the required number of instructions (in assembler code) versus the energy, and the number of transmissions versus the energy, an estimation of both the energy per instruction and the energy per byte is made based upon the performance of a TelosB sensor platform, with the MSP430F1611 as MCU and the CC2420 as radio chip. The power expenditure of both chips is expressed in number of microAmps (μA) in combination with a certain supplied Voltage. The unit for power is Watt and can be expressed in AmpèreVolt (AV), but also in Joules per second. Therefore, the energy per instruction can be calculated according to the following equation:

$$\frac{Joules}{instructions} = \frac{Joules/second}{instructions/seconds} = \frac{Watt}{instructions/second} \tag{5.1}$$

However, the datasheet does not specify the required power to execute a certain instruction, instead, the power to execute an instruction cycle is defined. The duration of instructions is expressed in number of instruction cycles, that is, the number of periods the master clock needs to progress, and can vary between one and six cycles per instruction for the MSP430 instruction set. Therefore, the energy to execute an instruction is expressed in this chapter as the energy to execute an instruction cycle times the average of the number of instruction cycles per instruction.

The duration of an instruction depends on the type of instruction, i.e., does it require none, a single or two operands, and the employed addressing mode. The MSP430 has seven addressing modes, which determine how the instruction operand should be interpreted by the instruction operator. The operand could specify a literal value, a memory address, which would result in a memory fetch at the specified address, a register, etc. Depending on the addressing mode or combination of addressing modes, the specific instruction requires more or fewer cycles. In order to obtain an estimate of the average number of cycles required per instruction, a disassembled program that performs both data processing and data transmission is parsed. During the parsing of the program, both the number of instructions and the total sum of instruction cycles is collected. The final results enable to obtain an estimate of the average number of cycles per instruction, which is 2.5 cycles per instruction. Note that loops are not taken into account since it is just an estimation, but thanks to the size of the analyzed program, the average number of cycles per instruction will not deviate significantly when considering loops.

The calculations are based upon a master clock, that is the clock determining the instruction cycle duration, running at 4 MHz. According to the specifications of the MSP430, the power consumption is equal to 500 µA per MHz, which results in a total current of 2 mA, when supplied with a voltage source of 3 V. The power consumption is therefore 6 mW. As a result, the energy per instruction cycle (*epi*) is formulated in Equation 5.2.

$$epi = 6mW/4MHz = 0.00375\mu J/cycle \tag{5.2}$$

The required energy to execute a series of instructions, in function of the number of instructions per Byte, nI, and the number of bytes that need to be processed, nB, can therefore be defined by Equation 5.3

$$expended\_energy\_instructions = 2.5 \times nI \times nB \times epi \tag{5.3}$$

The expended current when transmitting data at the maximum transmission power is specified to be 17.4 µA when supplied with a voltage source of 3 V. The power consumption of transmitting data is therefore 52.2 mW. The energy expenditure of the radio transmission is expressed in the energy per transmitted byte. Since the CC2420 has a maximum transmission rate of 256 Kbits per second, i.e., 32 Kbytes per second, the energy per byte (*epb*) is formulated in Equation 5.4.

$$epb = 52.2mW/32\ KBytes\ per\ second = 1.63\mu J/Byte \tag{5.4}$$

Besides the power required to transmit the data, the number of instruction cycles required to enable the transfer of the data from the MCU towards the radio chip over the SPI bus is also considered. Analysis of the code section responsible for executing the SPI data transfer led to the conclusion that there is a section which needs to be executed once, the size of 574 instruction cycles, and a section that needs to be executed for each byte transfer, the size of 150 cycles. The estimated energy consumption of the radio in function of the number of bytes that need to be transmitted, nB, is therefore expressed as Equation 5.5.

$$\begin{aligned} expended\_energy\_radio &= (574 + nB \times 150) \times epi + nB \times epb \\ &= 574 \times epi + (150 \times epi + epb) \times nB \end{aligned} \tag{5.5}$$

In order to determine the boundary on the number of instructions that justify a complex data processing over the raw transmission of data, it is required that the outcome of both Equation 5.3 and Equation 5.5 is equal, the energy expenditure of both the instruction execution and the radio transmission is equal. Such function determines the lower boundary of the number of required instructions per byte to account for a raw data transmission. If the number of instructions per byte is higher than this boundary, the raw transmission of data is more effective than processing the data. When requiring that both equations need to be equal, Equation 5.6 arises, where the number of instructions per byte are depicted in function of the number of bytes that either need to be processed or transmitted. Note that this equation does not take into account the transmission of the compressed data, which will eventually need to be transmitted, therefore, the equation is favorable for the compression of data.

$$\begin{gathered} 574 \times epi + (150 \times epi + epb) \times nB = 2.5 \times nI \times nB \times epi \\ \Updownarrow \\ nI = \frac{574}{2.5 \times nB} + \frac{(150 \times epi + epb)}{2.5 \times epi} \end{gathered} \tag{5.6}$$

Figure 5.1 depicts the number of instructions per byte in function of the number of bytes to be processed or transmitted, where the expended energy is equal for both cases. It is clear that sending the raw data is more efficient in terms of energy when the number of instructions required to process a single byte exceeds 237 assembler instructions. This statement is true for a minimum number of bytes of 50 bytes that need to be processed. For a small amount of data, it is more efficient to process the data and send just the result. Note that this is merely a coarse estimation and does not take into account the expended energy of the transmission of the preprocessed data. However, the results provide a guideline and certainly an upper bound on the number of instructions where data processing is justified regarding the energy efficiency.

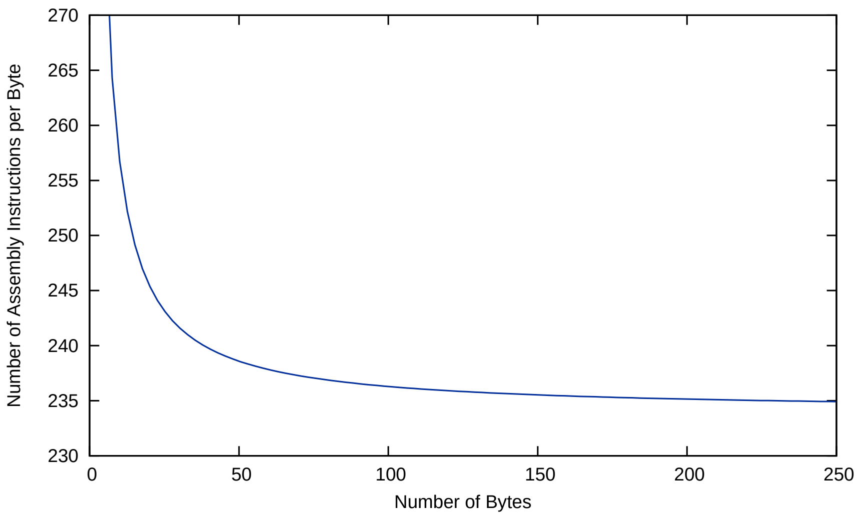


Figure 5.1: *Required number of instructions to account for raw data transmission*

5.3 Related work

Unlike Chapter 3, where an overview of available WSN related MAC protocols is provided, this section discusses work that is relevant to the employed methods in this chapter, which is not necessarily related to WSNs. The first subsection handles research related to bandwidth and rate distribution. The Fair Scheduling MAC protocol takes the bandwidth requirements of all nodes into account. As a consequence, the sensor nodes need to share the available bandwidth in a fair manner. Therefore, it is wise to take notice of the basic works regarding fair queuing, even though the methods for fairly distributing bandwidth are not directly applicable to the allocation of slots. Fair can have multiple definitions, certain cases require a strict control on the expended bandwidth, where sensors are just assigned the requested bandwidth in order to maximize the available bandwidth for other sensors. In other cases, the fairness indicates the restriction of a single sensor to capture the resource such a long time that other sensors are unable to send information on time. The subsection on fair scheduling might overlap on some areas with the second subsection, which discusses well known real-time scheduling methods and the employment of real-time in Wireless Sensor Networks.

5.3.1 Bandwidth and Rate Distribution

A significant amount of research towards fair scheduling is performed in the years around 1990. One has to keep in mind that the congestion withing packet switched networks were already the focus of research for two decades at that time [125]. However, high speed networking provided challenges for the research community at that time regarding data traffic control [126]. One of those challenges was congestion, which was the focus of most of the fair scheduling protocols that were conceived around that time. Note that the high speed networks that are referred to, are wired networks, where in most cases the congestion occurs within the router.

One of the earlier fair queuing (FQ) algorithms is proposed in [127], which focused on the congestion problem for the infinite storage case. Previously, most works on congestion management had been targeting buffer management in order to prevent congestion, whereas the author of this work considers an infinite storage space and proposes a fair queuing concept in order to make the network behave more fairly. Note that the method was already popular in the operating system world, but is claimed to be applied for the first time in packet switched networks. The work proposes to ensure the fairness of a switch with multiple inputs and outputs by providing an input queue for each possible network source and serving the queues in a round-robin manner, instead of having just a single first-in, first-out queue. As a result, any badly behaving node will only obtain a portion of the bandwidth inversely proportional to the number of nodes that are actively sending packets to the switch.

However, a downside of the protocol is that when packets of a different size are handled, one source is assigned a higher bandwidth than other sources, since the round-robin is done on a per packet base. Moreover, not every source in the network necessarily requires the same amount of bandwidth, the successful delivery of a packet is not determined by means of the same performance metrics for all traffic classes. The work in [128] recognizes this, as it states that the high speed networks are expected to be used for a wide range of traffic classes, each having its own performance requirements. The work considers two types of connections: rate controlled and best effort. During the connection setup phase of a rate controlled connection, a user requests a required average service rate and jitter bound. An accepted connection is ensured of receiving its required bandwidth and jitter bounds. The best effort connections are not guaranteed any bandwidth, but an effort is made to provide the lowest possible delay and loss probability. Two algorithms are proposed to achieve these goals: a rate-based scheduling and Hierarchical Round Robin Scheduling. The former queues its packets for transmission in frames of fixed size, transmits the entire queue and then proceeds to the next frame. The rate control is achieved by queuing the arriving packets according to a specific algorithm into the transmission frames. For each connection, four parameters are maintained, which identify the number of packets that can be sent per frame in order to comply to the bandwidth demand, the current frame in which packets of this connection are stored, the number of packets that have already been stored and how many frames to skip should the number of stored packets exceed the number of allowed packets for this connection. As such, the scheduling of incoming packets is merely a simple summation in order to determine in which frame the packet should be queued. Should any frame have too few packets to fill an entire frame, best effort packets are used to complete the frame. The hierarchical round robin method also considers frames of fixed size, but these frames are split in fixed slots. The number of slots a connection id receives, depends on its negotiated bandwidth. In order to differentiate between the different rates, the HRR has a hierarchy of service lists, where the topmost list the shortest frame length has and is therefore used to serve the highest service rate. Interesting to note is that the requested bandwidth is partitioned between service lists by allocating a fraction of the slots. In order to interleave the operation of the different levels, the upper level is operating, and serving cells in a round robin fashion, until a predefined boundary and then yields its control, such that level two can serve its cells. This protocol could be said to show a lot of similarities, at least conceptually, to the protocol that is discussed in this chapter. However, HRR does not determine the order in which connections are being served, the protocol does not clearly define how the bandwidth is split between the different service levels, the calculation of the boundary which determines how many cells should be processed is not specified, resulting in a protocol which is perfectly suited for rate control, but not for slot allocation where an absolute guarantee needs to be given that no collisions will occur, even when a connection drops or is added to the system.

Like HRR [128], is the Stop-and-Go queuing framework, proposed in [125], a non-work-conserving rate controlled service discipline, which does not allow a connection to send packets at a higher rate [129]. Non-work-conserving means that even if packets are available for transmission and no other work needs to be done, the scheduler will nevertheless wait until the appropriate time to send them. The goal of the Stop-and-Go protocol is to develop a framing strategy against congestion, that ensures a loss-free communication, a bounded end-to-end delay and is easy to implement, thereby reducing the required processing of the control function to a minimum. In order to achieve this, the protocol makes use of two policies: an admission control policy and a service discipline. Both policies make use of time frames, that are synchronized on a global network level. The admission policy ensures that the average bandwidth allocated to a certain source, is not exceeded during a single frame. Should any packet arrive that would violate this rule, the packet is not admitted until the following frame starts. Instead of sending all rate limited packets immediately to the next hop nodes in a FIFO manner, which would result in the formation of bursts, the packets are sent according to a stop-and-go queuing approach. This basically means that packets are sent during the time frame, after the one in which they were received. As a result, all packets coming from any given connection, experience an equal amount of delay, not taking into account a small jitter component, provided they travel over an equal number of hops. Since the size of the frame is the determinant factor when the total delay or required buffer size is considered, multiple frame sizes are proposed, where the smaller ones reduce the queuing delay and buffer size at the cost of a reduces flexibility in bandwidth allocation. Each frame size is a multiple of the previous frame size. During connection setup, the connection is classified and thereby associated according to one of the frame sizes, resulting in an admission policy which takes the respective frame size into account. The transmission of the packets is organized according to the frames, where packets residing in a smaller frame has a higher non-preemptive priority than larger frames. As a result, at the start of each frame, first the packets with the highest priority are sent, until the predetermined rate has been reached, which is enforced by the admission policy, and then packets of a lower priority are sent.

Whereas the previous two protocols enforced a strict control on the allowed packet rate, the protocol in [130][119], called Fair Queuing (FQ), does not agree on any maximum rate and allows a connection to send packets as any rate, provided that the bandwidth and buffer space is allocated in a fair manner to all available sources. It is claimed that a queuing algorithm determines the way in which packets from different sources interact with each other and this forms an essential component in the congestion control [119]. Moreover, such protocol should provide sufficient protection in the presence of maliciously behaving sources. The idea was to divide the bandwidth between the present sources equally. However, a round robin approach, such as in [127], fails to achieve this fairness due to possible variations in packet sizes. In order to resolve the issue, a bit-by-bit round-robin processing of the queues is proposed, where each data flow is assigned a queue. Since such processing is not feasible in reality, an approximation is made where complete packets are sent. However, the selection of which packet is to be sent next is based on the theoretical finishing time, should the bit-by-bit round-robin method have been used. The packet that ends the earliest is the next packet to be sent. As an answer to the diversity of networks where sources may need different bandwidths in order to provide the required performance, the authors add in a side note a weighted version of the FQ protocol, WFQ (Weighted Fair Queuing), that allows to adjust the respective weight of each of the flows, such that instead of each source receiving an equal amount of the available bandwidth, a flow is provided with a portion of the bandwidth respective to its associated weight. Note that even though the protocol is able to guarantee a throughput, irrespectively of the demands of the other flows, it is dependent on the number of flows, since each flow is assigned an either equal or proportional share of the throughput.

It is interesting to note that around the same time, at least two other protocols were designed that show similar features as (W)FQ. The first protocol is called VirtualClock, proposed in [126]. The target was to maintain an average transmission rate of data flows, while enforcing the agreed upon resource allocation for multiple priorities. On top of that, a security feature was installed to prevent malicious nodes to capture the entire bandwidth. The concept was derived from a Time Division Multiplexing (TDM) system, where interference could be entirely prevented thanks to the slot allocations, but which proved to be too rigid, resulting in a waste of bandwidth. The solution proved to be a virtual clock, which was assigned to each flow and ticked at every packet arrival from that flow at the rate of the inverse of the average bit rate. Each incoming packet is then timestamped with the current value of the virtual clock and is transmitted in ascending order of timestamp values. If packets have variable sizes, then the value used to increment the clock is set to be proportional to the packet size. Therefore, the protocol does not allocate slots as in a TDM system, but merely determines a transmission order. One of the major differences with FQ, is claimed to be the explicit resource reservation, allowing the nodes to specify the precise amount of bandwidth required for a successful operation of the network flow.

A second protocol that is similar to FQ and designed around the same time, is the packet-by-packet Generalized Processor Sharing protocol (GPS) [131] [132][133]. The focus of the work lies on the congestion control, where both the flexibility of the network and the performance guarantees are preserved. The network is supposed to support a wide range of transmission categories, while at the same time provide performance guarantees to real-time sessions. The work proposes the combined use of Leaky Bucket admission control [134] and a packet service discipline based on Generalized Processor Sharing in order to achieve this goal. The service discipline, GPS, is expected to allow users to demand a different performance, but it should not come at the cost of other users, so there needs to be certain fairness factor embedded in the protocol. The GPS protocol is a flow-based multiplexing discipline, which is work conserving and operates at a fixed rate r. If $S_i(\tau, t)$ indicates the amount of traffic from source i, that was serviced within an interval $(\tau, t]$, then for a GPS server the following condition (Equation 5.7) is true for any session i that is continuously backlogged in the interval$[\tau, t]$, and Φ_i represents a positive real number, that indicates the relative weight towards other input streams.

$$\frac{S_i(\tau, t)}{S_j(\tau, t)} \geq \frac{\Phi_i}{\Phi_j} \qquad \textit{where } j = 1, 2, \ldots, N \tag{5.7}$$

Each session i is guaranteed a rate equal to Equation 5.8

$$g_i = \frac{\Phi_i}{\sum_j \Phi_j} r \tag{5.8}$$

One of the interesting properties of GPS, is that it is able to provide queuing delay guarantees when the incoming data stream is rate limited by means of leaky buckets.

Since the GPS protocol is, like the bit-by-bit round-robin method in FQ [119], a theoretical abstraction and does not transmit packets as a whole, a packet-by-packet Generalized Processor Sharing protocol (PGPS) [131] [132][133] is proposed , that represents an approximation of GPS, where entire packets are sent. The concept is identical to the one employed in [119], where the order in which packets are processed is according to their theoretical finishing time of GPS.

An application of WFQ is described in [135], where the algorithm is used to design a new type of scheduler, the Earliest Eligible Virtual Deadline First scheduler (EEVDF). The idea is to design a flexible and accurate algorithm to allocate resources in an operating system. Every client needs to send a request, indicating the duration it needs. Based on the associated weight of the client and on the already allocated service time, each request is assigned an eligible time, that is, the start time in terms of WFQ, and a deadline, i.e., the finishing time in terms of WFQ. Both parameters are expressed in a virtual time, which is introduced to track the progress in an ideal fluid-flow system. A request can only be processed if it becomes eligible, that is, its eligible time is lower than or equal to the current virtual time. Like in WFQ, the request with the earliest deadline is next to be scheduled. The client share, $f_i(t)$, at time t is defined in Equation 5.9, where A is the set of all clients active at time t and w_i is the weight assigned to client i.

$$f_i(t) = \frac{w_i}{\sum_{j \in A(t)} w_j} \tag{5.9}$$

The time that can be used by the client in the ideal fluid-flow system, provided the client share remains constant during a time interval $[t, t + \Delta t]$, is equal to $f_i(t)\Delta t$ virtual time units. Note that the definition of the virtual time in the publication, depicted in Equation 5.10, is also dependent on the active clients, such that each client receives w_i real time units during one virtual time unit.

$$V(t) = \int_0^t \frac{1}{\sum_{j \in A(t)} w_j} d\tau \tag{5.10}$$

The implementation of the scheduler makes use of a time quantum, that is, the smallest time duration the scheduler can assign to a task, such that the overhead of the scheduling and context switching is not greater than the execution of the task itself. In other words, instead of the fluid-flow model, a discrete time base is used. The assignment of the allocated time to a task is therefore in units of time quantum, which does not necessarily match with the allocated share according to the fluid-flow model. If a client is assigned too much time, i.e., the time quantum is larger than the required time, the client releases its allocated time quantum upon completion in favor of some other task to be scheduled. A request can be scheduled when the service time the client should receive in the fluid-flow model equals the service time it actually has received. Therefore, when a client has received too much than its fair share, it needs to wait. If a client has received fewer than its fair share, its request can be scheduled immediately.

The contribution in [136] consists of the introduction of an ideal Fair Service Curve link-sharing model, where all service curves are satisfied, the excess bandwidth is fairly distributed among the service classes and a priority service is provided. A scheduler is proposed which makes an attempt to implement this ideal model by guaranteeing the service curves of all leaf classes, and trying to satisfy the service curves of the other classes and fairly distribute the excess bandwidth. The model allows to analyze the performance of link-sharing where a certain resource needs to be shared by multiple classes. For example, the available bandwidth can be shared between a video streaming application and an FTP transfer. While the video streaming requires a minimum bandwidth, the most important parameter is the delay. For the FTP application the delay is irrelevant. In the proposed model, the delay can be decoupled from the bandwidth by providing a non-linear service curve. A compromise is that not all service curves can always be guaranteed. Therefore, the scheduler operates in two modes, if all service curves can be satisfied, then the link-sharing algorithm is allowed to operate. Otherwise, the real-time class takes over the operation, which schedules the packets with the lowest deadline. The same Fair Service Curve is also used to provide a bursty best effort service in [137] and [138]

In contrast to the previously discussed algorithms, a Hierarchical Round-Robin algorithm (H-RR), proposed in [139], does not focus on a fair bandwidth distribution, but a low latency for high-rate sessions. The protocol introduces a tree in which each node is assigned a weight, according which a proportional rate is allocated. A leaf node represents a session, while the other nodes are intended to share their available bandwidth over the present children. The position of a session is selected carefully such that its requested or required rate can be satisfied. The closer the session is located to the root node, the higher its rate. While operating, the root node is assigned service slots, that is, slots large enough to accommodate the transmission of a single packet. Each of the children of the root node is assigned a service slot in a Round-Robin manner. If the child is a leaf node, then the packet is transmitted, otherwise, the service slot is assigned to one of the children of the node in a similar manner as the root node assigns service slots to its children. The resulting transmission schedule reflects the high rate services, which are repeated every cycle in case they are children of the root node. The services positioned one level lower are not handled every cycle, just a single service, which is determined in a Round-Robin manner, is handled per cycle.

Whereas the previously discussed protocols were more focused on bandwidth and rate allocation in a general manner, the following protocols consider the available bandwidth as a resource that can be fragmented. One of the discussed protocols in Chapter 3, [59], makes use of location information to determine the transmission opportunities. When multiple nodes are located within the same area, nodes are assigned the same slot in alternating frames, which is also a form of bandwidth sharing.

The proposal, MRBS and MRCS, in [140], is directed to bandwidth allocation of variable rate transmissions for a wideband CDMA systems with different traffic patterns. The same protocol, only with the focus on a generic wireless network, is discussed in [141]. The wideband CDMA (WCDMA) standardization from 3GPP that is used as a reference for the discussed protocol is named Universal Mobile Telecommunications System (UMTS). Two of the operational modes, included in the standard, are Direct Spread (DS) CDMA, and Time Division Duplex (TDD) CDMA. These modes are based on the UMTS Terrestrial Radio Access Frequency Division Duplex (UTRA FDD) and the UMTS Terrestrial Radio Access Time Division Duplex (UTRA TDD) proposal respectively. The discussion is based on both UTRA models. The goal is to optimize the number of users within a given bandwidth. All users therefore need to send a request containing an estimation of the required resources.

The multirate operation is a key feature in 3G wireless networks, since it enables different types of services, i.e., different data rates. In DS-CDMA systems, the spreading factor is kept constant. The resulting range of available data rates is fixed, for example, the supported transmission rates in UTRA FDD are defined by $2^i \times 15kbps$, where $i = 0, 1, \ldots 7$. Selecting a data rate, sufficiently large to support the requested rate can lead to a wastage of available bandwidth. A Hybrid time division multiple access (TDMA/CDMA) technique defines different slot sizes for different transmission rates. The low rate services are assigned short time slots, while the high rate services are allocated large slots. While the proposal allows a flexible bit rate assignment, it is not trivial to envisage a large number of different slot sizes to allow for a rate that is close enough to the requested rate. Moreover, the number of services that need to be considered might deviate from the number of specific slot sizes. At a certain point, there might be such an amount of small rate services such that there are not anymore sufficiently small slots.

To alleviate the previously discussed issues, the proposed algorithm, Most Regular Binary Sequence (MRBS), makes use of equally sized slots, which it allocates to the different services according to their demands. Bandwidth is allocated, such that it matches as close as possible to the requested bandwidth. A request, which can be expressed as a rational fraction of the total available bandwidth, can employ this type of schedule. Three properties are associated to the schedule. First, the resulting asymptotic rate is equal to the requested rate. Second, the scheduling sequence is periodic and deterministic and third, the slots are evenly spaced over the periodic cycle. The MRBS generating function is defined in Equation 5.11, where n is a nonnegative integer, representing the slot index, $0 < p \leq 1$, representing the fraction of the total bandwidth. The generated MRBS is cyclic and deterministic.

$$s(n) = \Big\lceil (n+1)p \Big\rceil - \Big\lceil np \Big\rceil \tag{5.11}$$

Figure 5.2 illustrates the theoretical requested bandwidth for a request equal to one third of the total available bandwidth, i.e., $p = 1/3$, represented by the solid blue line. Both functions that constitute the MRBS generator are also depicted in the figure, $f_1(n) = \lceil np \rceil$ represented by the green dashed line and $f_2(n) = \lceil (n+1)p \rceil$ represented by the dotted purple line. The difference between $f_2(n)$ and $f_1(n)$ results in a repetitive sequence $\{1, 0, 0\}$, since both functions increase by one every three slots, but are shifted by a single slot.

The MRBS pattern is used to allocate equally sized slots to the requesting users. A $'1'$ in the sequence indicates the permission to use the specified slot. Since the resulting pattern is directly related and closely matched to the requested rate, such schedule assures a minimal delay and buffer occupation. While the proposed schedule is elegant and provides a repetitive schedule which matches the requested bandwidth as close as possible, sequences of different users may try to allocate the same

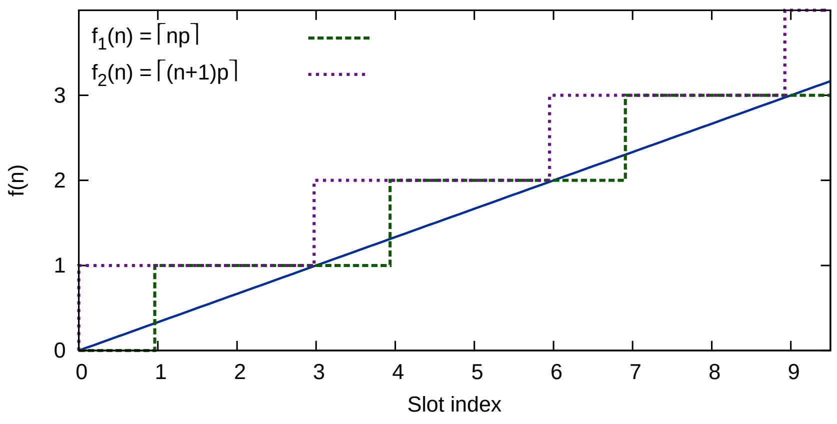


Figure 5.2: *f_1 and f_2 of MRBS with asymptotic mean equal to* $1/3$

slot. The protocol includes a conflict resolution algorithm, such that if a conflict should arise, the algorithm is able to resolve it.

The work also proposes an improvement of MRBS, that is, Most Regular Code Sequence (MRCS), which is more flexible in its use for different spreading factors and has a smaller variance than MRBS. The generation function is depicted in Equation 5.12, where n and p represent the slot index and the requested fraction of the available bandwidth. The variables p_U and p_L are both of the form 2^{-i} and represent the highest value lower than p, called lower-closest code (LCC) rate, and the lowest value higher than p, called upper-closest code (UCC) rate, respectively. All parameters should comply to $0 \leq p_L < p \leq p_U \leq 1$. The resulting pattern allows to switch between the LCC and UCC rate, thereby achieving a plethora of possible combinations to match the requested bandwidth. Like the MRBS, the MRCS also requires a contention resolution mechanism.

$$c(n) = (p_U - p_L) \times \left\lceil \frac{(n+1)p}{(p_U - p_L)} \right\rceil - \left\lceil \frac{np}{(p_U - p_L)} \right\rceil \tag{5.12}$$

While to protocol proposed in [142] is not related to Wireless Sensor Networks, the methodology of the protocol in order to generate a collision-free centralized schedule is sufficiently interesting to be included in this discussion. The protocol is proposed within the context of IEEE Std 802.16, which has been formed to recommend an interface for Fixed Broadband Wireless Access (FBWA) systems that can support multimedia services. The wireless mesh network employs a TDMA scheme with fixed-length time slots, where the slots for transmissions should be scheduled in a conflict-free manner for the entire neighborhood of the sender. The standard specifies that a new node, also called a subscriber station, joining the network should first listen to transmitted mesh configuration messages, such that a coarse synchronization is obtained. Based on the overheard messages, a neighbor list is constructed from which a sponsoring node is selected. A sponsoring node can be considered as the node which relays MAC messages to and from the base station for the newly joined node. The proposed protocol considers fairness, channel utilization and transmission delay as performance metrics. The standard defines three scheduling methods, amongst which the centralized scheduling is used in the proposed work. The base station gathers traffic demands from all subscriber stations, determines a schedule such that no collisions occur and informs all nodes about the schedule. By means of service tokens, a schedule is generated which complies to all demands. These tokens ensure that the number of slots, which need to be allocated to a specific node, is proportional to its traffic demand. As a result, fairness between the nodes is guaranteed. Thanks to the direct relation between the traffic demand and the slot scheduling, the duration of the cycle can be reduced. Such feat can be achieved by dividing each of the traffic demands by the greatest common divisor of all traffic demands of the nodes in the neighborhood. Based on the scheduling tree and the set of service tokens from all nodes, a scheduling matrix can be generated by means of an algorithm. The algorithm selects a link from the set of links and allocates slots for the link, while marking all conflicting neighboring links as interfered. As a result of the slot allocation, the service token of the transmitter is decreased by one and the service token of the receiver is increased by one. The process is repeated for other links until all service tokens of the considered nodes have decreased to zero.

A protocol designed for video conferencing, [143], discloses a method for reserving fractionally schedulable resources. Instead of considering a resource as being either busy or available, the resources are considered to be fractionally schedulable. The bandwidth of a particular resource may only be partially used during the reserved time. Therefore, a scheduling engine is proposed for fractionally scheduling resources. The engine uses a resource usage table in order to fix the resource allocation. A resource request is granted and added to the table if the resource usage does not exceed the total capacity of the network.

Interesting to note is that most of the discussed algortihms operate on the higher network layers, where they count on the lower network layers to resolve collisions etc. The protocol that is being proposed and discussed in this chapter, however, is operating on the MAC layer and is responsible for the collision avoidance and if necessary, collision resolution.

5.3.2 Real-Time in WSN

While the previous section targeted algorithms specifically for the scheduling and distribution of bandwidth, this section discusses protocols that schedule resources such that a deterministic behavior with regard to transmissions can be observed, that is, a real-time behavior.

The algorithm proposed in [144][145] investigates the delay jitter characteristics and bounding for real-time channels in a packet-switched store-and-forward wide-area network with general topology. The scheme consists of three parts: an establishment, a scheduling and a rate control mechanism. During the establishment phase, clients, that request a real-time service, are required to provide traffic characteristics and performance requirements during the channel establishment. Those criteria are matched to the capacity of the nodes on the establishment path. Each node is required to verify whether the acceptance of the new channel does not jeopardize the guarantees provided to previously established channels. All nodes have two types of traffic queues, one for regular traffic and one for real-time traffic, where all traffic is ordered according to the packet deadlines. The latter has a higher priority and is scheduled according to a variant of the Earliest Due-Date Scheduling scheme.

The rate control regulates both packets that arrive at a higher rate than predicted and the jitter between packet arrivals. Assuming a packet arrives at its predetermined time t_0, then it will be assigned a deadline equal to $t_0 + d_{i,n}$, where $d_{i,n}$ represents the local delay bound to channel i in node n. The associated eligible time, that is, the time upon which the packet becomes eligible for transmission, is in that case equal to $t_0 + d_{i,n} - J_{i,n}$, where $J_{i,n}$ is the local jitter bound assigned to channel i in node n. If a packet arrives on time, the value of $J_{i,n}$ is equal to $d_{i,n}$ and therefore, the packet becomes eligible upon its arrival. While the difference between the eligible time and the deadline of a packet remains constant, the eligible time can be increased when a packet arrives faster than expected, thereby shifting the first transmission opportunity for the packet. Thanks to the combination of the employed methods, the algorithm achieves a constant delay over the whole transmission path, with only the jitter introduced by the last hop in the path.

Since CSMA/CD might incur unpredictable message delays due to the collisions, a specific type of channel sharing is introduced in [146]. Instead of a CSMA type of access, the discussion targets a TDMA type of access. The hard real-time processes are assigned a complete slot, thereby ensuring a predictable delay, without any collisions. However, assigning a full TDMA slot to the soft real-time processes, would result in an inefficient use of the available resources. Time slots should be dimensioned to cope with the largest packet possible, which automatically leads to unused bandwidth in all other cases. Moreover, if a process is inactive or its transmission queue is empty, its associated slot remains reserved, while being unused. Therefore, the concept of a shared channel is defined, i.e., a slot which can be used by a set of processes. Access to such channels needs to be scheduled, such that collisions are avoided. No specific scheduler is proposed, since a wide range of real-time communication methods for multiple-access networks can be applied. An example is provided how the properties of one of the access methods of IEEE Std 802.11, that is, DCF, can be used to provide such access.

The algorithm, proposed in [147], provides hard real-time guarantees in the context of Wireless Sensor Networks. The network is organized according to cells, where Frequency Division Multiplexing (FDM) is used to prevent conflicts among neighboring cells. All nodes constituting a single cell operate at the same frequency and are able to communicate with all nodes in the cell. Router nodes, which are able to communicate at two different frequencies at the same time, are introduced in order to enable inter-cell communication. The algorithm focuses on the intra-cell communication, that is, the communication within a cell. The time is divided into slots, that is, TDMA is used to avoid collisions. The algorithm is specifically designed for periodic hard real-time messages, where the periodicity of the messages is exploited to avoid conflicts. All nodes in a cell have knowledge regarding the periodicity of the messages of the other nodes. The message transmission, and therefore also the node selection, is based on the Earliest Deadline First (EDF) scheduling. Since all nodes are aware of the other nodes messages and periodicity, and are synchronized, the scheduling mechanism can be performed in a distributed manner. If a node has no more data to be sent, it includes such information in the last packet it sends, which is captured by the nodes in the cell. The node which is next to be scheduled can use this extra time to already start transmitting its packets. The protocol is not dependent upon the reception of such last packet, it is merely an optimization.

The following article [148] claims to provide a real-time MAC for Wireless Sensor Networks, called Virtual TDMA for Sensors (VTS). The network is organized into cells or clusters, where the number of nodes comprising a single cell determines the superframe length of the TDMA cycle. The nodes in a cell are unaware of any superframe, since their communication and synchronization method is based upon S-MAC [22]. A node sends out a control packet, which is called a sync packet in S-MAC, and then waits until N_C slots have passed until sending again data. N_C represents the local variable containing the number of neighbor nodes. A new node that is willing to join the current cell is required to contend for the slot. If it wins the contention, it can send in the slot. Since all nodes are aware of the messages being sent, they can keep their local variable N_C up to date, ensuring they adhere to the superframe length. It is claimed that providing a single slot for each node is more throughput efficient and provides an upper boundary to the latency. However, the adaptiveness of the protocol results in potential contention issues, such as described in Section 5.1. While the protocol ensures a deterministic delay while the network is stable, that is, there is no change in the number of nodes in a cell, it can not be classified as being a real-time MAC. The number of slots, and therefore also the delay, is dependent on the number of nodes per cell, which can be deemed unpredictable due to for example connection losses, battery failures or node mobility.

The work presented in [149], RT-link, is targeted towards real-time wireless communication in industrial control, surveillance and inventory tracking. The algorithm employs a fixed frame duration, which is composed in a similar manner as IEEE Std 802.15.4 [150] of a number of Scheduled Slots, i.e., only nodes that are assigned such slot are allowed to transmit, and a number of Contention Slots, where all nodes can contend for access to the medium. The beaconing mechanism, which triggers the synchronization mechanism, is implemented by means of an additional AM receiver for indoor operation, where an AM transmitter periodically emits a beacon, or an atomic clock receiver for outdoor operation. During initialization, a node tries to synchronize to the beacon signal. After a successful synchronization, a node contends for access in one of the randomly chosen Contention Slots by skipping the duration of the Scheduled Slots. A HELLO message is sent in such Contention Slot, which is then forwarded to the central gateway, whose task it is to assign a Scheduled Slot to the node. The scheduling of a Scheduled Slot is done by means of a method for which knowledge of the global topology is required. The neighbor lists of all nodes are taken into account to construct a connectivity and interference map. Based upon these data sets, the slots of the nodes can be scheduled based on $k - hop$ coloring, such that when two nodes are assigned the same slot, the nodes are separated by at least $k + 1$ hops. On top of this coloring, a search is initiated for a minimum delay schedule, which is similar to the distance-two graph coloring problem. The end result is a schedule where all data coming from the leaf nodes reaches the gateway during a single TDMA cycle. The schedule also indicates at which rate the nodes are allowed to send, that is, it specifies whether nodes are allowed to send every frame, every two frames, every four frames, etc. by means of a four bit rate index. The former procedure is only valid for fixed nodes. Mobile nodes, on the other hand, are never scheduled due to their mobility. They only use the Contention Slots by listening to neighboring fixed nodes and synchronizing temporarily by taking into account the slot number of the overheard message. In such manner, the start of the frame can be deduced and therefore also the start of the Contention Slots, in which the mobile nodes contend for access. The algorithm is implemented on the Nano-RK real-time Operating System, where every slot within the TDMA frame is scheduled as a periodic task.

A Low-power real-time MAC protocol (LPRT) is proposed in [151], which disregards network topologies based on multi-hop transfer methods, due to the latency and throughput decimation that such methods impose. Instead, a star topology is proposed, which can be extended by introducing multiple base stations, thereby creating a clustered network. Like in the previously discussed protocol, a fixed frame length is adhered to and the frame is split into a number of slots defining a contention period (CP) and a number of slots that constitute a contention free period (CFP). The CP is used to allow stations to associate with the base station, as well as to forward Non-real-time asynchronous traffic. A frame starts with a beacon transmission, initiated by the base station, which contains information regarding resource grants (RG), acknowledgements for packet arrivals during the last period, etc. Based upon the RG, the nodes are able to identify in which slot during the CFP either a transmission or reception is scheduled for them. An issue with such resource allocation mechanism can arise when the link quality starts to drop, thereby causing the loss of some beacon packets, such as also demonstrated in [152][153], which are both improvements of the LPRT protocol. In the latest improvement, the resource grant mechanism is adjusted such that the grant is not limited anymore to the current cycle, instead it is valid until mentioned otherwise. However, while the protocol targets real-time communication, the scheduling of the slots is not disclosed, making it hard to determine whether real-time operation is supported.

A protocol called real-time MAC [154], based upon S-MAC [22], is claimed to provide real-time properties. Its goal is to remove the unpredictable latencies, accumulated due to the unpredictable channel access times by the CSMA/CA medium access method. The proposed method provides a solution for networks in which a single communication is present in the entire network, that is, there is only a single sink and a single source. In order to alleviate the uncertain latency issue, real-time MAC makes use of a feedback control packet, called Clear Channel. The packet ensures that the two-hop neighbors are prevented to transmit data, while further located nodes are granted access to the medium. This protocol ensures a minimum latency pipeline, where the next packet is not transmitted until the network is ready to accept it without risking interference to packets further up the pipeline.

A considerable number of works, such as [61][64], consider alternative wakeup schedules, where the wakeup periods of all nodes from the source to destination are shifted. Such schedule ensures a reduced packet latency thanks to the ability of a packet to traverse the complete tree in a single cycle, instead of a single hop per cycle. Such skewed wakeup scheduling is also applied in [155], where the observation is made that concurrent transmission from multiple children is disregarded in the design of a schedule which is based on the level of the children. Therefore, the proposed algorithm, SPEED-MAC, includes an extra step during which a short SIGNAL message is sent to the parent before actually sending the data. The parent is able to detect whether multiple children tried to send a SIGNAL packet at the same time, since the signal signature of a collision has a regular and consistent signal strength higher than the normal noise level. The parent forwards a SIGNAL packet to its parent, wherein an indication is placed whether or not a collision occurred. Since the children are able to overhear this packet transmission, they all have an idea whether or not a collision occurred. When only a single child tried to access the medium, the regular slotted transmission is applied. When multiple sources tried to access the medium, SPEED-MAC employs a variation of CSMA/CA as in S-MAC [22]. The protocol methodology requires the number of data slots and signal slots to be equal to the depth of the routing tree. Such demand requires a stable network operation, where a (semi-)fixed topology is expected. On the other hand, thanks to the stability, the packet latency can be predicted accurately in the case of a single source. Due to the possibility of having multiple sources transmitting concurrently, an additional uncertainty is introduced, making the protocol only suitable for soft real-time communications.

A work that is focused on Wireless Sensor Networks in a hard real-time context is proposed in [156]. The protocol makes use of both pipelining and data aggregation to provide a better packet latency, while at the same time reducing the energy consumption. The assumed network topology is a tree, where all sensors collect data and forward it towards the sink node. To reduce the required time to forward data and prevent interfering transmissions, multi-frequency access based on parent relationship clustering is employed. Neighboring clusters are assigned different frequencies to eliminate the possibility of interference. The frequency assignment to the nodes is performed in several steps. First, Dijkstra's algorithm is employed to find the shortest path from each sensor to the sink. Each parent encountered is assigned a different reception frequency, thereby implicitly creating clusters consisting of a parent and its children. Second, the number of frequencies is reduced by means of graph coloring, while in the third step the number of frequencies is yet to be reduced to the number of available channels. This last step is performed by fusing clusters of the same level. A last step introduces a TDM between the nodes of the same cluster, where each node is assigned a different slot. The optimization of data aggregation and pipelining is achieved by aggregating the received packets of the children and send such aggregated information during a single slot to its own parent. The pipelining is achieved by having a skewed sleep schedule of parent and children, such that the reception of the parent is synchronized with the transmission of the children. While the parent is transmitting its information to its own parent, its children are sleeping. It is claimed that such protocol design allows to ensure a predictable packet delivery latency from sensor to the sink and therefore provides guarantees for a hard real-time context. Such statement can be considered to be true for a specific network setup, which remains stable. However, the latency is a function of the network construction, that is, the number of levels in the tree, the number of nodes in a cluster, the number of frequencies, the number of slots, etc.

5.4 GCD slot allocation

This section proposes a novel protocol, which initiates the discussion of a protocol which allows a fair scheduling of slots. The proposed protocol is a conceptual design where a possible method to allocate slots to a sensor node is disclosed. The approach provides the foundation for the protocol depicted in Section 5.5. The operation of the protocol has some things in common with [142], even though the related work does not consider a Wireless Sensor Network. Both works require the nodes to announce their traffic demands, also called the requested bandwidth, to nodes which are assigned to relay messages to the base station, which is called the sink in the case of WSNs. Moreover, in both works a common factor is sought in order to reduce the scheduling length, which is the greatest common divisor (gcd) between all requested bandwidths. It can also be claimed that the algorithms both assign a sufficient number of slots, such that the demand for bandwidth is supplemented on average. However, whereas the related work is a centralized scheduling protocol, which requires a full overview of the section of the network it is responsible of, the protocol discussed in this section and its improvement, discussed in the following sections, are distributed scheduling protocols. Provided the relay nodes, also called parent nodes, have sufficient bandwidth to assign the requested bandwidth, no intervention from the sink is required. It is the responsibility of the parent nodes to verify whether sufficient unused bandwidth is available and request for more bandwidth to their respective parents when required. Note that the synchronization and joining of a node to the network is out of scope of this discussion. Similar methods such as discussed in [142] and Section 4.8 can be utilized to perform the synchronization and coordinate the joining of nodes to the network. A trivial extension of the network association procedure, that is, the joining of the network, allows nodes to inform their immediate parents of their bandwidth requirements.

Unlike the related work, the requested bandwidth is converted to a fractional value representing the number of slots, or partial slots, required per frame to fulfill the bandwidth demand. A bandwidth requirement which is lower than the bandwidth provided by a single slot is approximated by allocating an integer number of full slots in n frames over a total of m frames, where $n < m$. For example, a bandwidth request of 300 bytes per frame, where the slot size is equal to 400 bytes can be achieved by assigning a full slot during three frames, while the fourth frame some other node may use the specific slot.

One of the considerations of the protocol is the undisrupted and stable operation of the network. As such, the protocol makes use of a bandwidth allocation notification, which is directed specifically towards a single node. The notification contains the slot indexes and the frames in which the node is allowed to use the slot. The slot assignment forms a repetitive pattern and repeats itself after a number of frames, thereby ensuring that the schedule needs to be sent only once and does not disrupt the already allocated slots. The number of frames that form a repetitive pattern is called a cycle. One of the challenges to this approach is the dynamic behavior of the network, where nodes are allowed to join and leave the network. In order to guarantee the stability of the network, this protocol utilizes a greatest common divisor (gcd) based slot assignment mechanism. All parent nodes maintain a list of children and their requested bandwidths, which are stored in a fractional notation. All fractions are reduced to lowest terms by calculating the gcd between the nominator and the denominator. Since the requested bandwidth of nodes is not deemed identical, the denominators of the fractions are most likely different. Note that a cycle of two frames where a node only transmits during a single slot and a frame of six frames, where the same node transmits during three slots results in the same throughput on average. By making this observation, it is clear that nodes can be scheduled according to their fractional bandwidth request, even when a node with a different request joins the network. The least common multiple (lcm) between the denominators of the fractions from all children needs to be found in order to ensure no interruptions in the execution of the already existing schedule. When all fractions have equal denominators, the nominator indicates the number of frames in which a slot is used, the denominator depicts the number of frames a cycle consists of and the integer part indicates the number of full slots are used each frame.

Unlike the protocol in Section 4.8, this approach allows to use a flexible number of nodes, provided that the total required bandwidth is lower than the available bandwidth. Besides the flexibility, the protocol also ensures a controlled utilization of the allocated bandwidth, ensuring that nodes are not assigned more bandwidth than required. If a greatest common divisor (gcd) can be found between the different bandwidth requests, which is divisible by the size of a slot, the fair scheduling of the requested data throughput is trivial such that the allocated slots are dispersed. This new approach allows for the maximization of the TDMA frame structure, thereby reducing the control overhead caused by synchronization and other control slots at the start of a frame. Moreover, the data slots can be enlarged, thereby reducing the overhead of the physical layer. Note that the length of a cycle is based on the lcm of the denominators. Such approach shows promising results when the requested bandwidths have a low lcm. However, for larger values, the cycle tends to become too large, thereby creating a repetitive sequence which is too long to be contained within the restricted space in the notification message. In order to remedy the issue, a more sophisticated approach, which makes use of a series of fractions, is proposed in the following sections. Since the following protocol is an enhancement of the current protocol, the performance analysis and implementation are solely related to the following protocol.

5.5 Fractional series based slot allocation

The goal of the newly proposed protocol discussed in this section is to provide a periodic slot allocation, such that the requested bandwidth is closely approximated. Like in the previous section, the slot allocation algorithm is based on the fractional representation of the requested bandwidth per frame versus the available bandwidth per slot. However, this algorithm makes use of a more refined method to determine the slot and frame access, thereby resolving the encountered issues of previous discussion. Note that the discussion in this section and the analysis in the following section are based on a theoretical model, which assumes a fluid flow, that is, the arrival of data is assumed to be continuous in function of time. Since in reality it is not possible to for example receive half a bit, the analysis on data arrival in bursts considers more practical data arrival flows and the implementation of the protocol on hardware takes into account the practical constraints. A similar methodology is used in [157], where a theoretical model is proposed which also assumes a fluid model. Afterwards the ideal model is relaxed during the implementation where realistic constraints are taken into account.

The algorithm, which is to determine a schedule that complies to the specified demands, needs to determine a common factor on which the repetitiveness of the schedule can be founded. The requested bandwidth is key in determining this common factor. This section first elaborates on the general methodology to find such common factor such that a periodic cycle is guaranteed. Besides the periodic cycle, this methodology also provides the necessary means to construct a schedule where the slot access of nodes is diffuse in time, without risking any slot allocation conflicts, not even when new assignments are made.

The requested bandwidth that is required to service the nodes can also be expressed as a fraction of the available bandwidth. In other words, the number of allocated slots is proportional to the total number of available slots according to this fraction. Every bandwidth request can be assigned a portion of the bandwidth in such manner. An unfortunate element is that the requested bandwidths are not necessarily correlated, that is, it can happen that they have no common divider, or in other words, the gcd (greatest common divider) of those bandwidths can be equal to one. As a result, the appraoch employed in the previous section can not be used. It might not be possible to deduce a repetitive pattern between the different slots, which is constrained by the number of slots within a frame. In order to force a common factor, all fractions of the available bandwidth could be approximated by a unit fraction with a denominator equal to a power of two. The value of the unit fraction should represent an equal or higher slice of the bandwidth than the value it represents. As such, a combination of the different fractions is easily achieved, while at the same time providing the opportunity for a dispersed slot allocation, without resulting into conflicts. New slot allocations can be fit in the remaining space within the existing schedule without disturbing the already allocated fractions. However, this type of approximation towards unit fractions results in an excessive quantization error, and therefore a waste of bandwidth. In order to minimize such error, that is, to estimate the requested bandwidth as closely as possible, the fraction is not represented by a single unit fraction, but by the sum of distinct unit fractions. Such sum is called an Egyptian Fraction [158][159], where for this specific case each of the unit fractions has a denominator equal to the power of two. As an example, a fractional bandwidth equal to $\frac{4}{10}$ can be approximated as the Egyptian Fraction $\frac{1}{4} + \frac{1}{8} + \frac{1}{32}$.

Interesting about this method is the split up of the requested bandwidth into higher and lower frequency parts. A sensor gets access to the resource at least in periodic intervals equal to the highest frequency. By interleaving the different fractions, a sensor is ensured access to the resource at regular intervals. Furthermore, the calculation of the approximation of the requested bandwidth needs to be performed only once, since further bandwidth requests will not interfere with the already allocated bandwidths. Moreover, since the unit fractions can be considered as frequencies, it is sufficient to specify the frequency and the initial starting point in order to deduce the remainder of the allocation pattern.

In order to clarify the concept, the following example will demonstrate a possible allocation procedure for the fraction $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32}$. The completed allocation is depicted in Figure 5.3, which shows the allocated slots for each of the unit fractions. The numbers in the slots represent the slot index modulo ten in order to improve the readability. The most restrictive fraction to fit, is the one with the highest frequency, which is $\frac{1}{2}$ in this case. The fraction uses the resource half the time in an interleaving way. The starting point for the fraction can be selected from two possible slots, slot 0 and slot 1. When converted to binary code, the slots would be identified as slot 00000 and slot 00001. In the figure, fraction $\frac{1}{2}$ is scheduled at slot 0, the rule is to follow the path with the lowest index, which designates an available fraction of $\frac{1}{2}$ at slot 1 for the other fractions.

The next most restrictive fraction is $\frac{1}{4}$. When no other fractions constrict its allocation, it can be scheduled with a starting point in slots 0 (00000), 1 (00001), 2 (00010) or 3 (00011) and an interval of four slots. Since fraction $\frac{1}{2}$ is already allocated to

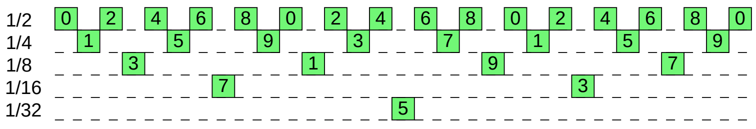


Figure 5.3: *Slot allocation for* $\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16}+\frac{1}{32}$

slot 0 and its interval is equal to two, both slot 0 and 2 are already allocated. The remaining slots that could provide a valid starting point for fraction $\frac{1}{4}$ are slots 1 (00001) and 3 (00011), of which the former is selected to allocate the fraction. The other fractions are scheduled in a similar manner, resulting in the slot assignment scheme depicted in the figure.

Note that thanks to the unit fractions having a denominator equal to a power of two, the starting point selection can be represented as the traversal of a binary search tree, such as depicted in Figure 5.4. The figure has marked the starting slots and allocated bandwidth for the previously discussed allocation example of fraction $\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16}+\frac{1}{32}$ by means of red circles and colored thick lines respectively. The binary codes on the right of the figure represents the slot index for each possible starting slot and can be used to trace back the followed path in the binary search tree. The use of a binary tree ensures that any additional fraction does not interfere with the already scheduled fractions, but it also ensures that the fractions are equally diffused over the available resources. The scheduling problem is therefore reduced to merely following the path in a binary search tree from left to right and checking whether the path is still free.

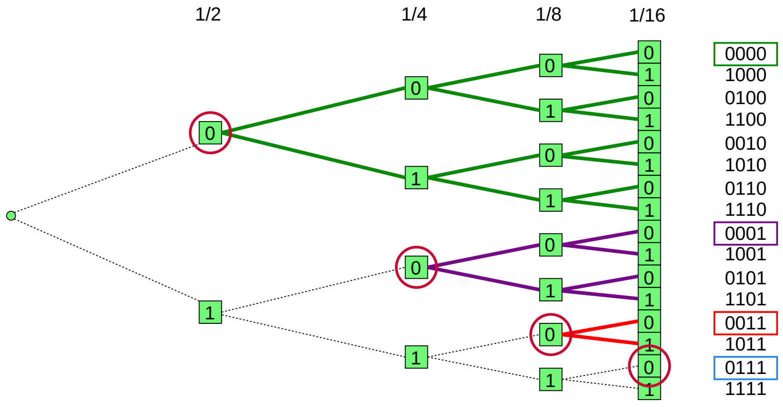


Figure 5.4: *Binary search tree*

While the above discussed approach is able to provide an efficient method to ensure a dispersed slot allocation without causing conflicts to previously allocated slots, its application is insufficient in heterogeneous networks where nodes might have a different bandwidth requirement. The previously discussed method assumes a minimum bandwidth requirement equal to a single slot, while in heterogeneous sensor networks some nodes might only need a fraction of the provided slot bandwidth. Therefore the following discussion handles the specific slot allocation mechanism that is to be used in a heterogeneous WSN.

Note that the requested bandwidth can be considered a fraction of the requested bandwidth per frame over the total available amount of resources per slot. Therefore, the requested bandwidth can be represented by a quotient and a remainder of this fraction, where the quotient represents the number of slots per frame and the remainder specifies a fraction of a slot. Moreover, the number of full slots can be considered as a fraction of the total number of slots in a frame. As such, there are two fractions that can be scheduled according to the previously disclosed methodology. When constraining the number of slots per frame to a power of two, the application of the method to the full slots fraction is straightforward. This single constraint ensures the quantization error is eliminated. As a result, any slot requirement can be expressed by an Egyptian Fraction and be scheduled accordingly, without the need to approximate the bandwidth request. The fractional slot request, that is, the remainder of the fraction of the bandwidth request over the available bandwidth per slot, needs to be processed in a different manner, since the request considers less bandwidth than a single slot provides.

As already mentioned in the introduction, dynamic slot sizes are not supported. However, a most accurate approximation of the requested bandwidth needs to be achieved, even while the request is lower than the available bandwidth per slot. By storing data in a local buffer, thereby aggregating the data into a single transmission buffer, the node is able to wait a number of frames until it has sufficient data to use a full slot. The methodology for the fractional slot requests is similar to the above described method. First a full slot is selected by traversing the binary search tree of the full slots. This slot is to be used as a shared resource by the different children. The fractional slot request is approximated by means of the previously described Egyptian Fraction, where the lowest possible denominator is constrained in order to prevent infinite or very long sequences. By means of a binary search tree, such as the one described above, the starting position of each unit fraction is determined. Whereas the starting position for the full slots indicated the slot index, here it indicates the frame index. The frequency of the unit fraction determines the periodicity and interval of the selected frames. Therefore, usage of the shared slot by the specific child is determined by the starting frame and interval of each unit fraction constituting the fractional slot request. As such, the frequency of the node to access the medium is at least equal to the highest frequency, that is, the highest unit fraction, of the Egyptian Fraction. On the other hand, the denominator of the lowest unit fraction determines the length of the cycle, which is the number of frames that form a periodic cycle. The lowest possible fraction, i.e., the precision, can be chosen at design time, depending on the requirements, as will be discussed in the following sections. Note that the shared slot can be used by siblings in frames where the node does not use the specific slot.

Note that the approximation of the bandwidth, which is expressed in two series of unit fractions, one for the quotient and one for the approximation of the remainder, can also be expressed as a series of n unit fractions depicting the number of required slots to supplicate for the requested bandwidth. As such, the approximation can be expressed as:

$$\begin{aligned} \frac{R}{S} &\approx \sum_{i=0}^{n} \frac{1}{2^{a_i}} \qquad (a_i \in \mathbb{Z} \text{ and } a_i < a_{i+1}) \\ &\text{or} \\ \frac{R}{S} &\approx \sum_{i=0}^{n} f_i \qquad (\text{with } f_i = \frac{1}{2^{a_i}} \text{ and } f_i > f_{i+1}) \end{aligned} \tag{5.13}$$

The parameters R and S represent the requested bandwidth and the slot size respectively. R is expressed as bytes per frame, while S is expressed as number of bytes. Therefore, the unit of the sum of fractions is $\frac{1}{\text{frames}}$, that is, every fraction signifies a frequency at which slots are scheduled. The values $a_0 \dots a_n$ denote the exponents of the denominators that are expressed as a power of two for each of the fractions, whereas $\frac{1}{f_i}$ denotes the interval, expressed in number of frames, between successive slot schedules according to fraction f_i.

While the start positions of the fractions was determined previously by traversing a binary search tree, it can also be expressed as an equation. Equation 5.14 depicts the start position, Fpos_k, of a fraction f_k, expressed as the offset relative to the start position of the first fraction, f_0.

$$\text{Fpos}_k = \begin{cases} 0 & (k = 0) \\ \displaystyle\sum_{i=0}^{k-1} \frac{1}{2} \frac{1}{f_i} & (k > 0) \end{cases} \tag{5.14}$$

The parameter f_i expresses the unit fractions within the approximation, sorted according to the denominator values in ascending order. Knowing that the unit of f_i is $\frac{1}{\text{frames}}$, the start position, Fpos_k, is expressed as the number of frames, implying that the value is a fraction of a frame if an allocated slot is not shared. The equation denotes that the offset is equal to half the sum of all periods of previous fractions. From this can be derived that the start position of fraction f_i occurs in the middle of the period of fraction f_{i-1}.

The allocation of the slots, that is, the start position assignment, is designed such that the accumulation of the remaining bandwidth does not become higher than two times the slot size. Every fraction results in the transmission of at least part of the data, thereby failing to transmit all accumulated data. Such failure to send all data can also be considered as a lack in bandwidth compared to the requested bandwidth. The difference between the already supplied bandwidth and the requested bandwidth is defined as the remaining bandwidth R_i, which is for each fraction different. The bandwidth is expressed as number of bytes per time unit, which can also be considered as the rate of the incoming data that needs to be transmitted. Therefore, the accumulated data during the interval determined by the specific fraction f_i is equal to:

$$R_i \frac{1}{f_i}$$

At each periodic interval of fraction f_i, data the size of a full slot S is transmitted, thereby reducing the remaining accumulated data. As a result, the remaining accumulated data after each scheduled slot that is allocated according to a fraction f_i equals:

$$R_i \frac{1}{f_i} - S$$

Since the fraction indicates the frequency per frame, the remaining bandwidth per frame due to the imprecision of fraction f_i is therefore equal to:

$$R_{i+1} = f_i \left(R_i \frac{1}{f_i} - S \right) \tag{5.15}$$

Note that in the special case of fraction f_0, the remaining bandwidth R_0 equals the requested bandwidth R.

The Equation 5.15) can be simplified by eliminating its dependency on its lower term remaining bandwidth. It is known that:

$$R_{i+1} = f_i \left(R_i \frac{1}{f_i} - S \right) \qquad \textit{and} \qquad R_i = f_{i-1} \left(R_{i-1} \frac{1}{f_{i-1}} - S \right)$$

Therefore, substitution of R_i in the equation for R_{i+1} leads to the following equation:

$$R_{i+1} = f_i \left(\frac{f_{i-1}}{f_i} \left(\frac{R_{i-1}}{f_{i-1}} - S \right) - S \right)$$

Substituting all R_i, with $0 < i$ results in:

$$R_{i+1} = f_i \left(\frac{f_{i-1}}{f_i} \left(\cdots \frac{f_0}{f_1} \left(\frac{R_0}{f_0} - S \right) - S \cdots \right) - S \right)$$

By rewriting the previous equation and taking into consideration that R_0 equals to R, the following equation can be obtained:

$$R_{i+1} = R - f_i S - f_{i-1} S \cdots - f_1 S - f_0 S = R - \sum_{j=0}^{i} f_j S \tag{5.16}$$

It is claimed that the slot assignment ensures that the amount of data arriving during the slot scheduling interval of a fraction f_i at a rate of R_i, the remaining bandwidth, is not larger than two times the slot size. That is, it is claimed that:

$$R_i < 2 f_i S \qquad \text{or} \qquad \frac{R_i}{f_i} < 2S \tag{5.17}$$

with R_i being the remaining bandwidth per frame (bytes per frame), S denotes the slot size and $f_i = \frac{1}{2^{a_i}}$ indicates the unit fraction. The proof of this statement is provided here.

Proof:
The statement is proven by means of induction.

Step 1:
Since the approximation consists solely of unit fractions with a denominator equal to the power of two and the scheduling algorithm of the slots according to the individual fractions is applied to a list of unit fractions sorted in ascending order of denominator values. Therefore, the following expression is true when $a_i < a_{i+1}$:

$$\sum_{i=1}^{n} \frac{1}{2^{a_i}} < \frac{1}{2^{a_0}} \qquad (a_i \in \mathbb{N} \text{ and } a_i < a_{i+1}) \tag{5.18}$$

Note that the approximation of the requested bandwidth can be expressed as:

$$\frac{R}{S} = \sum_{i=0}^{n} f_i \qquad (\text{with } f_i = \frac{1}{2^{a_i}} \text{ and } f_i > f_{i+1}) \tag{5.19}$$

The combination of both equations, Equation 5.18 and Equation 5.19, gives:

$$\frac{R}{S} - f_0 < f_0 \qquad (\text{with } f_0 = \frac{1}{2^{a_0}})$$

$$\Updownarrow$$

$$R\frac{1}{f_0} < 2S \tag{5.20}$$

The remaining bandwidth, R_0, is equal to the requested bandwidth, R, since f_0 is the first fraction. Thereby is proven that for a fraction f_0 the equation holds.

Step 2:
In this step, the proof will indicate that the Equation 5.17 holds for a fraction f_{k+1} if it is assumed that the equation holds for a fraction f_k. It is therefore assumed that:

$$\frac{R_k}{f_k} < 2S \tag{5.21}$$

Considering the remaining bandwidth, R_{k+1}, of the individual fraction, the accumulated data during the interval determined by the slot schedule, allocated according to fraction f_{k+1} is equal to:

$$\frac{R_{k+1}}{f_{k+1}}$$

From the remaining bandwidth equation (Equation 5.16), it is known that:

$$R_{k+1} = R - \sum_{i=0}^{k} f_i S$$

Substitution of the variable leads to the following equation:

$$\frac{R_{k+1}}{f_{k+1}} = \frac{1}{f_{k+1}} \left(R - \sum_{i=0}^{k} f_i S \right) \tag{5.22}$$

From Equation (5.13), it is known that:

$$\frac{R}{S} \approx \sum_{i=0}^{n} f_i$$

which can also be formulated as:

$$\frac{R}{S} - \sum_{i=0}^{k} f_i \approx \sum_{i=k+1}^{n} f_i \qquad (\text{with } \ 0 < k < n) \tag{5.23}$$

Since f_i equals to $\frac{1}{2^{a_i}}$, with $a_i < a_{i+1}$, it is known that:

$$f_{k+1} > \sum_{i=k+2}^{n} f_i \tag{5.24}$$

Since the approximation of fraction R/S either ensures an exact match or an even higher bandwidth, the combination of Equation (5.23) and Equation (5.24) provides:

$$\frac{R}{S} - \sum_{i=0}^{k} f_i < 2 f_{k+1}$$

which can also be written as:

$$\frac{1}{f_{k+1}} \left(R - \sum_{i=0}^{k} f_i S \right) < 2S \tag{5.25}$$

From Equation (5.22) and Equation (5.25), it can be concluded that:

$$\frac{R_{k+1}}{f_{k+1}} < 2S$$

Q.E.D.

This proof shows that during the time between the scheduled slots that are allocated according to the same fraction, the size accumulation of the received data will remain below the threshold of the capacity of two slots. As a result, after the scheduling of the slot, the size of the remaining data, that arrived in that interval, will be lower than the capacity of a single slot. This statement is key to the performance of the protocol. The approximation is composed out of a series of unit fractions, where each fraction supplicates part of the requested bandwidth. Each fraction, which has a lower frequency than any of the previous fractions, is used to maintain the remaining bandwidth within bounds, such that by scheduling slots according to lower fractions, the requested bandwidth can be supplied. This would not be possible if the received data during a single interval, determined by one of the fractions, would exceed this limit of two times the capacity of a slot.

Note also that according to the start position algorithm, Equation 5.14, the position of each subsequent fraction f_i is placed in the middle of the interval determined by the previous fraction f_{i-1}. Hence, the accumulated amount of data arriving according to the remaining bandwidth, R_{i-1}, at the moment a slot is scheduled according to fraction f_i, is half the amount of data that arrives during a period $\frac{1}{f_{i-1}}$, that is:

$$\frac{1}{2}\frac{1}{f_{i-1}}R_{i-1}$$

From Equation 5.17 it is known that this amount is smaller than the slot size S. Hence, the scheduling of a slot according to fraction f_i results in a decrease of the buffer size.

To clarify the employed methods an example of a slot allocation is provided. In this fictive example, a frame consists of four slots, each with a bandwidth of four bytes per second. The requested bandwidth is equal to nine bytes per second, which is equivalent to $9/16^{th}$ of the total bandwidth. In order to obtain the required number of slots per frame, the requested bandwidth is represented as a fraction over the available bandwidth per slot, i.e., 9/4. The approximation of the number of slots is therefore expressed as $2 + \frac{1}{4}$. The number of full slots is represented as a fraction of the total available number of slots, that is, $\frac{2}{4}$. The Egyptian Fraction representation consists of a single unit fraction, $\frac{1}{2}$ and according to Equation 5.14, slot 0 is available to be used as initial slot. Based on the periodic character of the fraction, a slot is assigned to this node every two slots according to this fraction $\frac{1}{2}$.

The currently scheduled bandwidth is equal to $\frac{8}{16}$, which is insufficient. In order to schedule the remaining $\frac{1}{16}$, first a shared slot needs to be determined by traversing the binary tree or the equation. The slot that complies to the requirements is slot 1, as $\frac{1}{2}\frac{1}{f_0} = 1$, with f_0 being $\frac{1}{2}$. The scheduling of the fraction $\frac{1}{4}$ requires either the traversal of the binary fraction allocation tree, which is a different tree than the slot allocation tree, or by means of the equation, which shows that frame index 0 is the first allocation that can be used for this fraction. Two cycles of the final allocation result is shown in Figure 5.5, where the frame indexes are depicted at the top of the figure and the slot indexes at the bottom. It can be noticed that the fraction $\frac{1}{2}$ is scheduled starting from slot 0 and then every two slots for each frame, while the fractional slot is scheduled in slot 1 in frames $0, 4, 8, \ldots$ which results in a perfect fit of the requested rate of $9/16^{th}$ of the total available bandwidth.

Thanks to the Egyptian Fraction notation with a denominator equal to a power

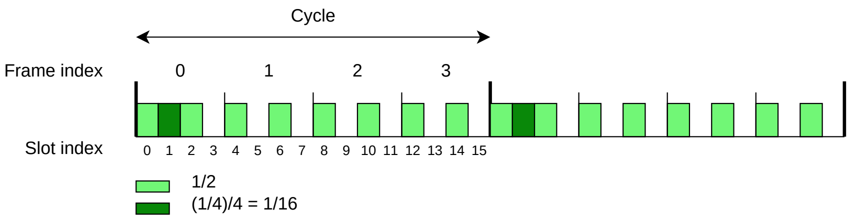


Figure 5.5: *Allocation of 9/16 in a 4 slot period*

of two, the slot assignment for both the full slots and the fractional slots can be represented by binary code in a very efficient manner. By depicting each unit fraction in terms of the lowest unit fraction, only a few bytes are required to encode this information. For example, if the precision of the fractions is 128 (the lowest possible fraction is 1/128), the sum of fractions $1/4 + 1/32 + 1/164$ can be expressed as $(128 * 1/4) + (128 * 1/32) + (128 * 1/64)$, or simplified as $32 + 4 + 2$. The binary representation of the sum is 0010 0110. Only a single byte is required to represent the unit fractions that are comprised in the approximation. Each fraction is associated with an initial slot and frame index. The size of the slot id is determined by the number of slots within a frame, while the size of the frame index is constrained by the precision of the fractions, that is, the lowest possible unit fraction. When considering a frame consisting of eight slots and a fraction precision of 128, A single byte is required to represent the fractions, $3 + 7$ bits per fraction for the slot id and the frame respectively. With a precision of 128, a maximum of seven fractions can be obtained, which means $7 * 10 + 7$ bits, which is equal to 77 bits, this is 10 bytes to send a complete assignment information. A similar slot allocation encoding can be used to inform the children about the full slot allocation schedule.

Thanks to the representation of full slots by a series of unit fractions and considering each of them as a frequency, the discussed approach managed to create a diffuse slot assignment in a trivial manner, without causing conflicts in earlier allocated schedules. By approximating the remainder of the fraction of the requested bandwidth over the total available bandwidth per slot as an Egyptian Fraction, a similar method could be used to determine the frames in which a shared slot can be scheduled, such that a precise approximation is made of the requested bandwidth. The method makes use of data aggregation in order to store the gathered sensor information in a buffer, until the data can be sent in the allocated slot, providing a more efficient usage of the wireless medium. Besides the diffuse slot allocation and the elimination of need for a conflict resolution method, the employed method also reduces the control message overhead. A slot allocation needs to be sent only once, during setup time and is able to operate autonomously for the remainder of its lifetime, even while the remainder of the schedule might change significantly due to extra bandwidth requests from other nodes. Since the method requires data to be aggregated locally, this might have an impact on both the required local storage and the latency of the data transmission. The following sections analyze the performance of the protocol in more detail with regard to both parameters.

5.6 Theoretical Scheduling Analysis

This section performs an analysis of the performance of the proposed scheduling protocol from a theoretical viewpoint for both latency and buffer size. The latency is defined as the time between the arrival of the measurement data at the node and the transmission time of the packet containing this data. The buffer size is the size of the memory required to store the data sufficiently long to be transmitted in the next available slot. The analysis is considered to be theoretical, since the arrival of the sensor measurement data is assumed to be a linear function in time, similar to the bitwise round robin and GPS abstraction in [119] and [132]. However, thanks to the abstraction, some valuable results can be derived from the analysis, which are placed in the right perspective in the next sections, where a realistic data arrival pattern is considered.

In order to allow examples and numerical comparisons in the discussion, the available bandwidth of a single node is assumed to be 19200 bits per second, i.e., 2400 bytes per second. A single frame is assumed to have a duration of a second. The analysis considers for different number of slots the performance of the protocol for requested rates ranging from a single byte to 2400 bytes per second. For each of the requested rates a slot allocation is determined, of which the performance is determined by simulating the progress of time. The resulting buffer size and latency are determined based on the arrival time of the sensor measurements.

As the analysis will show, depending on the requested amount compared to the maximum capacity, there are cases that are easier to understand than others. The more trivial cases are those where the Egyptian Fraction notation consists of only a few unit fractions. The more unit fractions are needed to form the approximation, the more complex the analysis becomes. The discussion will first focus on a number of cases that are rather easy to comprehend. Later on, an extrapolation of the findings is done towards the more complicated approximations. In order to simplify the notation of the fractions, the bandwidth request is expressed as the number of required slots to supplicate it. For example, when a frame is divided in eight frames, each with a bandwidth of 300 bytes per second, the request of 77 bytes per second, that is, a bandwidth fraction of 77/2400, can be represented as $1/4 + 1/128$. The notation implies the usage of a single slot, which is shared by the two fractions. The first fraction makes use of the slot for $1/4^{th}$ of the time, while the other employs the slot for $1/128^{th}$ of the time. To convert this notation to the requested bandwidth, the fractions need to be divided by the total number of slots per frame.

5.6.1 fewer-term slot allocations

This section discusses initially the more trivial bandwidth requests which result in a Egyptian Fraction of maximum two terms, which will form an exact representation of the requested bandwidth, that is, no approximation is required. The requested rate fractions 77/2400 and 79/2400 are investigated which are reduced to $\frac{1}{4} + \frac{1}{128}$ and $\frac{1}{4} + \frac{1}{64}$ respectively. Fractions with only a single term are not considered in detail here, since they are too trivial to include in the discussion. The scheduled slot is periodic and provides sufficient bandwidth to service the requesting node, hence, the measured data only needs to be accumulated during the duration of the periodic interval and the maximum buffer size is equal to the slot size. Series with two terms are more interesting to analyze, since a single term is insufficient to service the bandwidth request. A second term at a lower frequency is required to send the remaining buffered data. The simulations in this section assume a frame with eight slots of equal size, that is, each slot has a capacity of 300 bytes and a duration of 125 ms. The approximation of the fractional slots is limited to 128, i.e., the lowest possible unit fraction has a denominator equal to 128. As a result, the maximum cycle is composed of 128 frames or 1024 slots.

Note that the performance of the protocol with regard to the latency and buffer size is influenced by the remaining data that has not been transmitted yet. Instead of sending data as soon as it is available, the approach aggregates measured data in a single buffer and allows to use the maximum slot bandwidth in order to maximize the transmission efficiency. At the start of the simulation, no such data is buffered yet, providing the impression of a better performance compared to when the protocol is already operating for a certain time. In order to portray the performance in a realistic manner, the analysis only considers the simulation information starting from slot 1024, that is, after the maximum cycle size has passed.

The performance analysis is based on the theoretical arrival of the measurement results from the sensor, compared to the flow of the packet transmissions. As previously noted, the theoretical flow is a linear function of the amount of measured data over time. In order to clarify the definitions of latency and buffer size in this chapter, Figure 5.6 depicts an example of both the theoretical and transmission flow. The latency is the maximum time that the incoming data needs to wait before being processed by the protocol. In other words, the latency is defined as the time between the theoretical and the transmission flow. The buffer size can be defined as the amount of data that needs to be stored, before a slot becomes available to send the data.

The operation of the protocol allows nodes to utilize the slot capacity maximally. The first analyzed bandwidth request is 77 bytes per frame, which is reduced to $\frac{1}{4} + \frac{1}{128}$, taking into account the slot size of 300 bytes. The fractional representation implies the usage of only a single slot which is shared in time. The slot, slot 0, is used for $1/4^{th} + 1/128^{th}$ of the time by this specific node. The scheduling algorithm provides that the initial frame of fraction $\frac{1}{4}$ is at frame zero and the initial frame of fraction $\frac{1}{128}$ is at frame two. Thus, the node has access to the resource according to fraction $\frac{1}{4}$ at frames $0, 4, 8, 12, \ldots$ and is allowed a transmission opportunity according to fraction $\frac{1}{128}$ at frames $2, 130, 258, \ldots$ Note that the scheduling of a slot according to the first term, fraction 1/4, provides a transmission rate which is just below the requested bandwidth. The sensor data which was not transmitted dur-

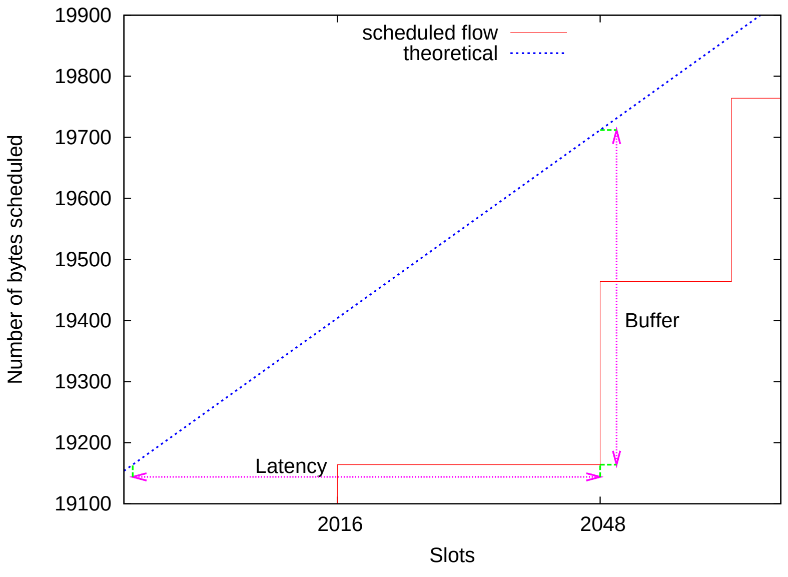


Figure 5.6: *Latency and Buffer size definitions*

ing the previous transmission opportunity is scheduled for transmission for the next slot, along with new sensor data. At every scheduled transmission according to the first term, the remaining measured sensor data accumulates, until the slot allocated according to the second term can be used to transmit the remaining data.

The operation of the protocol is depicted in Figure 5.7 for the requested bandwidth of 77 bytes per frame. The dotted blue line represents the theoretical linear function of the measured data arriving at 77 bytes per second, that is, the requested bandwidth, and the red line is the result of the data transmissions that have been scheduled at the designated slots. The step function of the transmissions indicates the employment of the full capacity of the slot, while the linear function of requested amount continues to accumulate data at a faster pace than the transmissions scheduled according to the first term. Such operation can be observed until slot 2064 (indicated in the small figure on the top left corner of the figure), which is the first slot of the 258th frame. This slot has been reserved for transmission according to the second term, that is, fraction $\frac{1}{128}$. These results show the intended operation of the protocol, there is a kind of periodicity in the behavior of the protocol. The cycle length of 1024 slots can be derived from the figure, which is the maximum allowed cycle length, based on the specified number of slots per frame and the constraint of 128 on the minimal term.

Since the allocation pattern results in a number of transmissions where an insufficient amount of data is transmitted, the buffer size will continue to increase until a slot allocated according to the following term of the series is encountered. Although it is clear from the previous discussion that the intended operation is depicted in

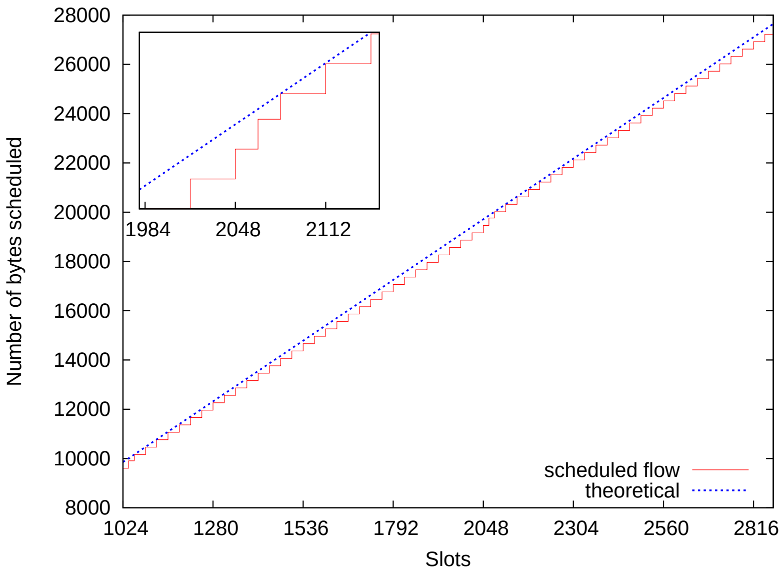


Figure 5.7: *Transmission pattern of 77 Bps with 300B slots* ($\frac{1}{4} + \frac{1}{128}$)

the results, to deduce the performance of the buffer size from the presented figure is not trivial. Therefore, it is depicted in Figure 5.8, which depicts the buffer size during the allocated slots for the request of 77 bytes, scheduled as $\frac{1}{4} + \frac{1}{128}$. The duration of a single frame is one second, therefore, the slot allocated according to fraction $\frac{1}{4}$ is scheduled every four seconds. Since the slot capacity is 300 bytes, every four seconds a transmission of 300 bytes can be executed, provided sufficient data is available. However, the requested bandwidth demands for an average transmission of 77 bytes per second, that is, 308 bytes per four seconds. Therefore, the slots allocated according to fraction $\frac{1}{4}$ do not provide sufficient capacity in order to meet the requirement of 77 bytes per second. Assuming the measured sensor data is being generated at a constant rate equal to the requested bandwidth, it can be said that more data is arriving than is being transmitted in the scheduled slots for fraction $\frac{1}{4}$. This clarifies the mechanics behind the increasing buffer size until the slot allocated according to fraction $\frac{1}{128}$ is scheduled. The scheduling of the last term resolves the difference between fraction $\frac{1}{4}$ and the theoretical linear rate function. As such, the lowest fraction determines the interval of the periodic behavior where the buffer size increases and decreases again.

As a comparison, the buffer size accumulation of the allocated slots according to the derived schedule for a resource request of 79 bytes per second is depicted in Fig. 5.9. The bandwidth request of 79 bytes can be approximated by $\frac{1}{4} + \frac{1}{64}$, indicating the scheduling of a single slot every four frames and an extra slot allocation every 64 frames. The results that are depicted in the figure confirm the influence of the lowest term on the periodic interval of the cycle. The interval has a period of 512 slots, which is equal to 64 times a frame of eight slots. Note that the maximum

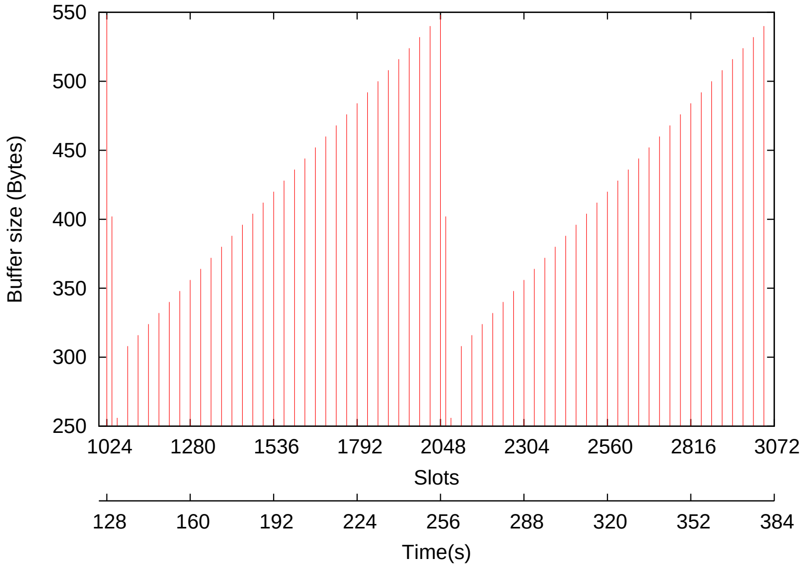


Figure 5.8: *Buffer size of 77 Bps with 300B slots ($\frac{1}{4} + \frac{1}{128}$)*

buffer size is smaller compared to the maximum buffer size of the 77 bytes per second schedule.

Based on the periodic behavior and the deterministic slot allocation method, it is possible to predict the expected buffer usage requirements. By using the example of a bandwidth request of 77 bytes per second, a method to determine the maximum buffer size can be derived. The request is represented as a series of fractions $\frac{1}{4} + \frac{1}{128}$. The fraction $\frac{1}{4}$ indicates the scheduling of a single slot every four seconds, that is, a transmission of 300 bytes every four seconds. Since the requested bandwidth relates to an average transmission of 308 bytes per four seconds, during each of the transmissions of a slot scheduled according to fraction $\frac{1}{4}$ the buffer size increases with eight bytes. Within the period of fraction $\frac{1}{128}$, the fraction $\frac{1}{4}$ is scheduled 32 times, of which 31 result in an increase of the buffer size (See Fig 5.8). Therefore, the accumulated data portion that has not been transmitted yet equals to 248 bytes (8 * 31). Note that on top of the excess data also a full slot of 300 bytes is generated by the sensor measurements during each period of fraction $\frac{1}{4}$. Therefore, the maximum buffer size is determined by both factors, that is, the slot size and the excess of generated data. As a result, the maximum buffer size is equal to 548 bytes. Note that this maximum value can also be found in Figure 5.8. Equation 5.26 depicts a means to determine the maximum buffer size for an approximation series with two terms, where f_0 is the frequency of the first fraction and f_1 the frequency of the second fraction, R represents the requested bandwidth in bytes per second, and S

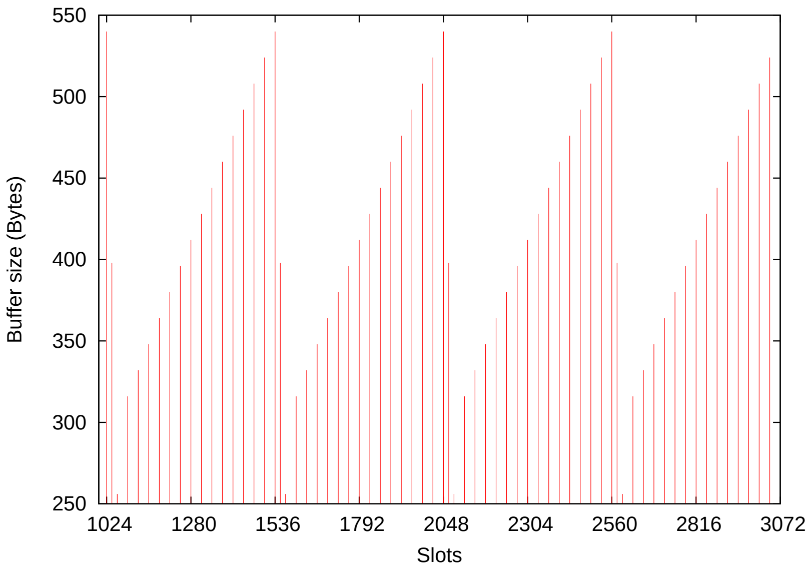


Figure 5.9: *Buffer size of 79 Bps with 300B slots ($\frac{1}{4} + \frac{1}{64}$)*

stands for the slot size.

$$Max\_buffer = S + \left(\frac{f_0}{f_1} - 1\right)\frac{1}{f_0}\left(R - f_0 S\right) \tag{5.26}$$

By employing the equation on the example of a bandwidth request of 77 bytes per second, where f_0 equals $\frac{1}{4}$, f_1 equals $\frac{1}{128}$, R equals 77 and S equals 300, the following equation would determine the maximum buffer size:

$$\begin{aligned} Max\_buffer &= 300 + \left(\frac{128}{4} - 1\right) * 4\left(77 - \frac{300}{4}\right) \\ &= 300 + 31 * 4 * 2 \\ &= 300 + 248 \\ &= 548 \end{aligned} \tag{5.27}$$

From the results it is clear that it is possible to calculate the maximum buffer size, based on the following parameters: the requested amount, the amount per slot and the Egyptian Fraction that approximates the requested amount. Note that this equation is only applicable to an approximation with two unit fractions. If only a single fraction is needed to approximate the requested amount, there is always sufficient capacity to transmit all data at once and thus the maximum buffer size is always smaller than or equal to the slot size. When more than one fraction is required to approximate the requested bandwidth, a more elaborate equation is required, taking into account all fractions within the approximation. The deduction of a more general equation to calculate the maximum bandwidth will be shown in the following section.

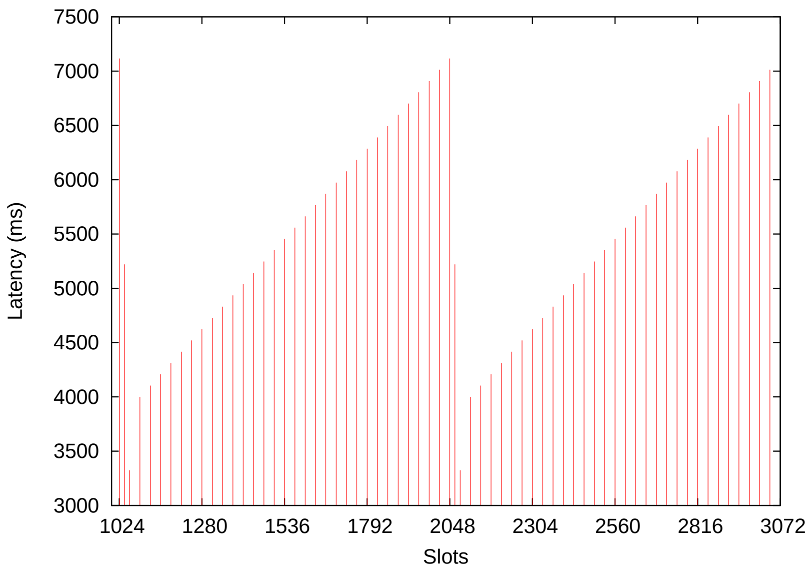


Figure 5.10: *Latency of 77 Bps with 300B slots* ($\frac{1}{4} + \frac{1}{128}$)

The second parameter to determine the performance of the protocol is the latency. The latency considers the first bit that was not yet transmitted and specifies the time it needs to be stored locally before being transmitted. The maximum latency is therefore in direct relation to the maximum buffer size, since the maximum latency can be considered as the time required to receive a number of bytes, equal to the maximum buffer size, at a rate conform the requested bandwidth. The direct relation can be noticed in Figure 5.10, which depicts the latency for each allocated slot with a bandwidth request of 77 bytes per second. Note the same behavior and periodic interval as the buffer size. When using the example of a bandwidth request of 77 bytes per second, the following deduction can be made to determine the maximum latency. It was already established that the maximum buffer size for this example is equal to 548 bytes. The time needed to receive this amount of data at a rate of 77 bytes per second is 7116,88 ms. Note the equivalence with the maximum latency depicted in the figure. Different bandwidth requests exhibit a similar behavior, such as for example the latency for a bandwidth request of 79 bytes per second that also has a similar curve of its latency as the curve of its buffer size. Interesting is the direct relation to the buffer size and the latency, since the maximum buffer size is deterministic. As such, the maximum experienced latency is also deterministic, resulting in an upper bound for the latency and a predictable transmission pattern.

5.6.2 Maximum buffer size for n-term slot allocations

Whereas the performance analysis in the previous section was limited to approximations of two terms, this section poses no limits to the number of terms. The general principle of the protocol is the periodic allocation of slots such that the resulting bandwidth is slightly too low to service the bandwidth request. The slot allocation according to the last frequency is capable of compensating the difference. In the previous section, only a single unit fraction was responsible for the bandwidth that is slightly lower than the requested bandwidth. In this section, any number of unit fractions might be combined to provide this bandwidth. Each of the fractions is responsible for a section of the bandwidth, resulting in a lower increase in the buffer size compared to when only the first fraction would be used. The derived equation for the maximum bandwidth therefore needs to be extended to a more general equation which takes into account every possible fraction within the approximation. The analysis starts with a comparable bandwidth and approximation, as the bandwidths discussed in the previous section, that is, a bandwidth request of 95 bytes per second which can be approximated by the series of fractions $\frac{1}{4} + \frac{1}{16} + \frac{1}{128}$. Afterwards, bandwidth requests with more terms are discussed. Like in the previous section, the frame is assumed to be composed of eight slots, each with a capacity of 300 bytes per second.

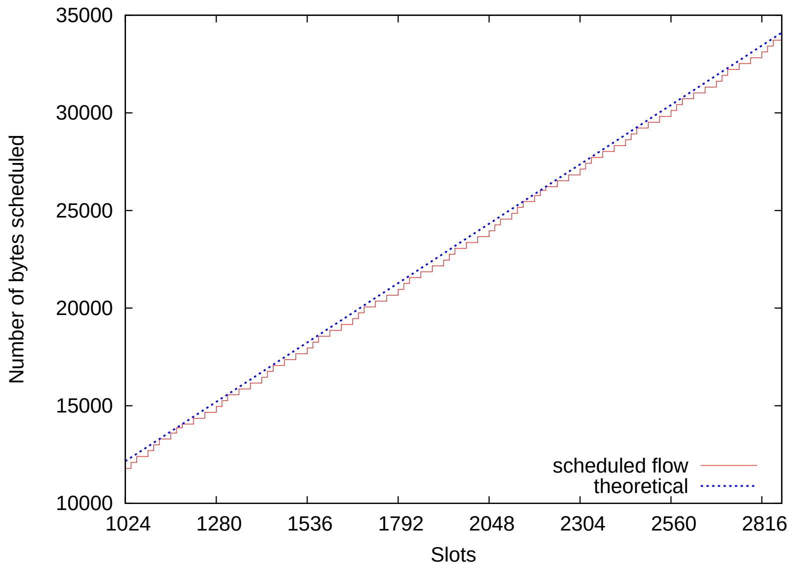


Figure 5.11: *Transmission pattern of 95Bps with 300B slots* ($\frac{1}{4} + \frac{1}{16} + \frac{1}{128}$)

The theoretical linear function of the average requested bandwidth and the step function representing the scheduled transmissions according to the protocol for a bandwidth request of 95 bytes per second are depicted in Figure 5.11. The figure indicates the more complex behavior of the protocol. The slots allocated according to the first term, i.e., unit fraction $\frac{1}{4}$, result in a bandwidth which is too small to transmit all received data. With each interval according to the first term, more and more data is accumulated in the buffer. By the slots allocated according to the second term, most of this excess data is transmitted. However, the second term still does not supply sufficient bandwidth, resulting again in an accumulation of the data in the buffer, which is emptied completely during the slot scheduled according to the last term.

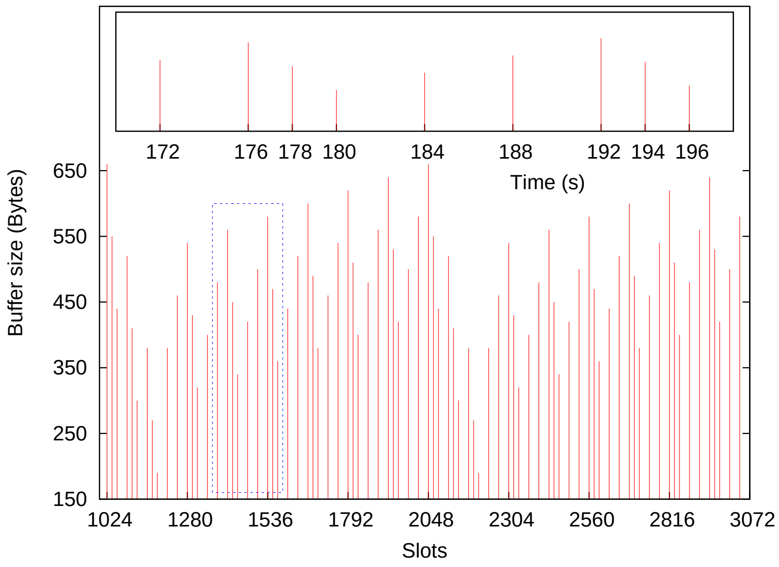


Figure 5.12: *Buffer size of 95Bps with 300B slots* ($\frac{1}{4} + \frac{1}{16} + \frac{1}{128}$)

The operation of the protocol for this approximation can be more clearly noticed in Figure 5.12, which depicts the buffer size for each of the allocated slots according to the approximation of the request of 95 bytes per second. The figure clearly shows the accumulation of the data in the buffer when only the transmission in the slots allocated according to the first term is considered. The transmission of data in the slot scheduled according to the second term reduces some of this excess data, however not entirely. In order to make this statement clear, a section indicated by a blue dotted rectangle is highlighted at the top of the figure. Since the fraction $\frac{1}{4}$ is scheduled every four seconds starting from frame 0 and a frame duration is equal to one second, the frames at $172, 176, 180, 184, 188, 192$ and 196 seconds are those frames where the shared slot is scheduled according to this fraction in the top figure. The slots that are allocated according to fraction $\frac{1}{16}$ have a periodic interval of 16 seconds, i.e., the frames at 178 and 194 seconds are selected to schedule the shared slot according to this fraction. Note that after the transmission in slot 1424 (frame 178), which is a slot allocated according to fraction $\frac{1}{16}$, a decrease in buffer size can be noticed. However, the bandwidth provided by the allocated slots of the second fraction is still not sufficient to compensate for the whole difference. This leads to an increase of the buffer size with a certain amount every period of the second fraction ($\frac{1}{16}$). The last fraction ($\frac{1}{128}$) is capable of compensating this difference and when it is scheduled, the buffer is emptied again. It is clear that the maximum buffer size depends on the subtle interaction between the allocated slots according to the different unit fractions.

The larger number of terms, especially for larger bandwidth requests, makes it harder to analyze the process of data accumulation and transmission step by step. Therefore, in order to derive a general equation that defines the maximum buffer size, the relation between the maximum buffer size of all bandwidth requests is investigated. As will be demonstrated in the following paragraphs, a recurring pattern can be discerned based on the gathered information, which enables the calculation of the maximum buffer size for any approximation, irrelevant of its number of terms.

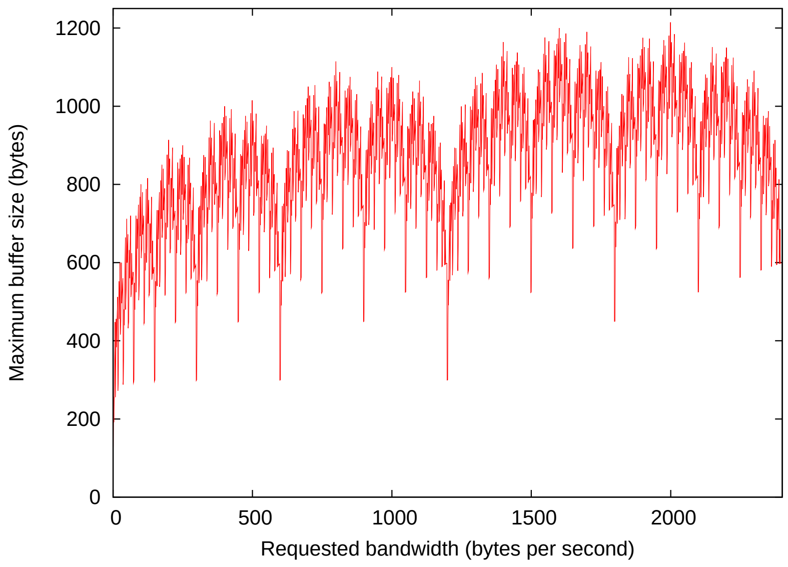


Figure 5.13: *Max buffer size vs bandwidth request for 300B slots*

Figure 5.13 depicts the maximum buffer size for all the possible integer bandwidth requests constrained by a maximal capacity of 2400 bytes per second. The results are based on a frame consisting of eight slots, each having a capacity of 300 bytes per second. The worst buffer size which might be needed is a little over four buffers, where each buffer is capable of offering sufficient space for a single slot, that is, 300 bytes. The bandwidth request which is responsible for the worst buffer size is a request of 1999 bytes per second, which would result in a buffer size of 1214 bytes, i.e., 4.04 buffers of 300 bytes.

Interesting to note is that the curve of the maximum buffer size contains some sections of which the pattern seems to be repeated, albeit not with the exact same values. The investigation of a small section of the gathered data, depicted in Figure 5.14 , where the requested rates are limited between 150 and 230 bytes per second , sheds some light on the discerned pattern. The full red lines in the figure indicate

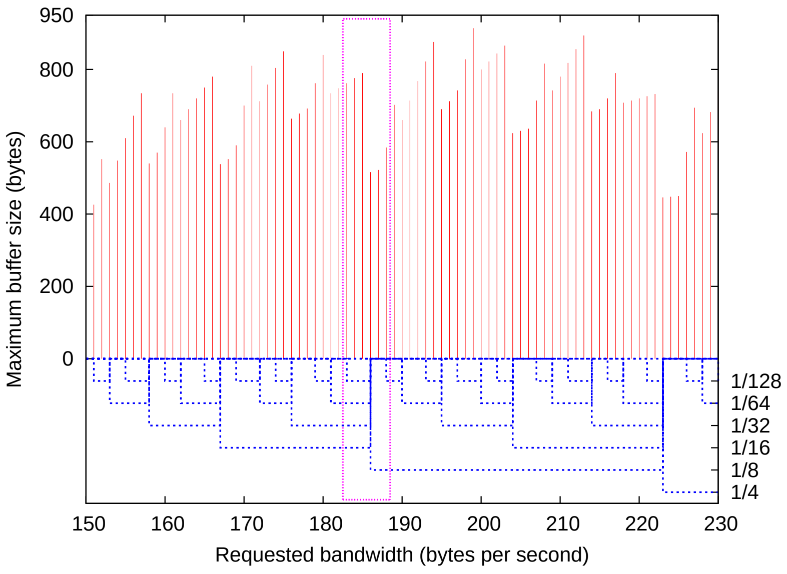


Figure 5.14: *Section of max buffer size vs bandwidth request for 300B slots*

the maximum buffer size encountered at the respective requested bandwidths. The dotted blue lines is the binary representation of the unit fractions that the approximation of the requested amount is composed of. The top blue line is the smallest fraction ($\frac{1}{128}$), the next blue line is the double of the fraction of the previous line and so on, until the bottom blue line is reached, which is fraction $\frac{1}{4}$. Note that the fraction $\frac{1}{2}$ is not shown here, since all the depicted bandwidth requests have this fraction in their approximation and would be shown as a straight line in the figure, which would not provide significant information. The composition of the approximations, that is, of which unit fractions the approximation consists, can be derived from these lines. For example, the requested bandwidth of 185 bytes per second is represented as $\frac{1}{2} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \frac{1}{128}$, while a bandwidth of 186 bytes per second is represented as $\frac{1}{2} + \frac{1}{8}$.

It can be noticed that the more fractions are used to approximate the requested bandwidth, the higher the maximum buffer size is. As can be expected, sequential bandwidth requests that can be serviced by the same approximation depict a certain increase in the maximum buffer size. Interesting to note is that the maximum buffer size increases in a more steep gradient if an extra fraction is added to the approximation. However, the main area of interest is the area where a sudden reduction of the maximum buffer size can be observed, which happens when a single larger fraction is used instead of a series of smaller unit fractions. This observation can be made for example by comparing the bandwidth request of 185 bytes (approximated as $\frac{1}{2} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \frac{1}{128}$) and the bandwidth request of 186 bytes (approximated as $\frac{1}{2} + \frac{1}{8}$), indicated by the dotted pink rectangle in the figure. These observations point out that each unit fraction of the approximation adds its own surplus to the maximum buffer size.

Based on these observations, it is possible to deduce a more general equation for the maximum buffer size that not only holds for approximations with a limited number of terms, but also for the more complicated ones. Equation 5.28 is the resulting equation that calculates the maximum buffer size that can be encountered when scheduling the requested bandwidth by means of the depicted protocol and when the reception of the sensor measurements is modeled as a linear function. Therefore, the maximal buffer size can be determined based on the following parameters: the requested bandwidth (R), the number of bytes per slot (S), and the Egyptian Fraction that approximates the requested bandwidth ($f_0 \dots f_n$). The equation is constructed by means of the same rationale as in Section 5.6.1 on which the following proof elaborates in detail in order to demonstrate the correctness of the equation.

$$\begin{aligned}
&if\ \ n = 0: \\
&Max\_buffer = R\frac{1}{f_0} \\
&if\ \ n > 0: \\
&Max\_buffer = R\frac{1}{f_0} + \sum_{i=1}^{n}\left(\frac{f_{i-1}}{f_i} - 2\right)\frac{1}{f_{i-1}}\left(R - \sum_{j=0}^{i-1} f_j S\right)
\end{aligned} \tag{5.28}$$

Proof:
The proof consists of three steps, during which first the statement is proven for an approximation consisting of a single fraction. The proof for an approximation composed out of two fractions is shown next, after which finally the proof for n fractions is provided.

Step 1: a single fraction
An approximation which consists of a single fraction f_0 ensures a slot allocation which has sufficient capacity to handle the transmission of the data according to the bandwidth requirements. Since only a single fraction is required to allocate slots, the accumulated data, collected during the interval between two subsequent transmissions, can be sent at once. Therefore no excess data is generated and the amount of accumulated data is equal to the maximum buffer size, which is equal to the duration, that is, the interval of the fraction, $\frac{1}{f_0}$, multiplied by the requested bandwidth, R:

$$Max\_buffer = \frac{1}{f_0}R$$

This shows that the Equation 5.28 holds for an approximation consisting of a single fraction.

Step 2: two fractions
An approximation which consists of two fractions implies that the slots of size S, allocated according to the first fraction, f_0, do not provide sufficient capacity to comply to the bandwidth requirement, R. After each transmission within a slot that is scheduled according to f_0, the amount of buffered data has increased with the excess amount of data, that is, the amount of accumulated data during the interval of fraction f_0 that has not been transmitted. The amount of remaining data after a period of $\frac{1}{f_0}$ can be expressed as:

$$Diff = \frac{1}{f_0}R - S \tag{5.29}$$

The increase in buffer size, due to the not yet transmitted amount of data, continues until a slot according to the next fraction is scheduled. During this period which is determined by fraction f_1, in total $\frac{f_0}{f_1}$ slots are scheduled, that are allocated according to fraction f_0. During $\frac{f_0}{f_1} - 1$ of the intervals between those slots the amount of accumulated data increases with *Diff*. The maximum buffer size is therefore equal to:

$$\begin{aligned}
Max\_buffer =& S + \left(\frac{f_0}{f_1} - 1\right) \text{Diff} \\
&= S + \left(\frac{f_0}{f_1} - 1\right)\left(\frac{1}{f_0}R - S\right) \\
&= S + \frac{f_0}{f_1}\frac{1}{f_0}R - \frac{f_0}{f_1}S - \frac{1}{f_0}R + S \\
&= \frac{1}{f_0}R + \left(2 - \frac{f_0}{f_1}\right)S + \frac{1}{f_0}\left(\frac{f_0}{f_1} - 2\right)R \\
&= \frac{1}{f_0}R + \left(\frac{f_0}{f_1} - 2\right)\frac{1}{f_0}\left(R - f_0S\right)
\end{aligned} \tag{5.30}$$

By means of this analysis, it is proven that Equation 5.28 holds for an approximation composed out of two fractions, $f_0 + f_1$.

Note that the maximum buffer size equation can also be formulated as follows:

$$\begin{aligned}
Max\_buffer =& S + \left(\frac{f_0}{f_1} - 1\right) \text{Diff} \\
&= S + \left(\frac{f_0}{f_1} - 1\right)\left(\frac{1}{f_0}R - S\right) \\
&= S + \frac{1}{f_1}R - \frac{f_0}{f_1}S - \frac{1}{f_0}R + S \\
&= \frac{1}{f_1}(R - f_0S) - \frac{1}{f_0}R + 2S
\end{aligned}$$

It is known from Equation 5.16 that the remaining bandwidth per frame R_1 equals:

$$R_1 = R - f_0S$$

Therefore, the equation can be written as:

$$Max\_buffer = \frac{1}{f_1} R_1 - \frac{1}{f_0} R + 2S \tag{5.31}$$

Note that $\frac{1}{f_1} R_1$ equals the accumulated data due to the remaining bandwidth per frame for a duration specified by fraction f_1.

Thanks to the operation of the initial slot positioning algorithm, the slot according to fraction f_1 is scheduled in the middle of the interval between slots scheduled according to fraction f_0, which results in the reception of data according to the remaining bandwidth for only half the time of a regular interval according to fraction f_0. This ensures a momentarily shorter interval between transmissions. The amount of data which is received according to the remaining bandwidth during the interval before the next transmission is scheduled is equal to:

$$\frac{1}{2} \frac{1}{f_0} R$$

Since the received data during the interval is lower than the slot size according to Equation 5.17, a decrease of the buffer size is experienced. Note that on both sides of the scheduled slot such decrease is encountered thanks to the reduced data arrival. The scheduling of the fraction f_1 is said to reduce the remaining bandwidth caused by the difference between the requested bandwidth and the bandwidth fraction f_0 is capable of providing. The reduction in buffer size can be called an effort to comply to the requested bandwidth and can be expressed as:

$$effort_1 = 2\left(\frac{1}{2} \frac{1}{f_0} R - S\right) \quad = \frac{1}{f_0} R - 2S \tag{5.32}$$

Based on Equation 5.31 and Equation 5.32, it can be said that the maximum buffer size is dependent on the remaining bandwidth for the duration specified by the fraction, and the effort required to reduce the accumulated data:

$$Max\_buffer = \frac{1}{f_1} R_1 - effort \tag{5.33}$$

As previously discussed, the method of approximating the requested bandwidth into a series of unit fractions is equivalent to splitting the requested bandwidth into a series of smaller bandwidths. Each fraction corresponds to a specific bandwidth and by allocating slots according to these fractions, part of the bandwidth requirement is fulfilled. When viewing the process as an elimination of bandwidth requirements, starting from the fraction with the highest frequency, then the remaining bandwidth after the scheduling of the slots according to a fraction, should be lower than the allocated bandwidth by this specific fraction. If this condition is not fulfilled, the remaining fractions, each of which allocates a lower bandwidth than its predecessor, will not be able to comply to the bandwidth requirements.

As a result, the remaining bandwidth, R_i, in combination with the duration of the slot scheduling interval, determined by the fraction f_i according which the slots are allocated, dictates the amount of data that could not be transmitted by the slot allocations according to previous fractions. A consequence of this approach is the increase of the buffer size during part of the interval due to the remaining bandwidth, while the other part, the effort, ensures a decrease in buffer size. Since the effort is a result of the scheduling of a slot according to the next fraction, the maximum buffer size is reached at the moment where data is to be transmitted by the last slot allocated according to fraction f_0 before a slot is scheduled according to fraction f_1.

Step 3: n fractions
In order to calculate the maximum buffer size, the approach from the previous step is employed, that is, each of the fractions is considered to comply partially to the requested bandwidth. Therefore, each of the fractions is partly responsible for the data accumulation, which is emphasized by the remaining bandwidth for a duration of the interval according to the fraction, and the effort to reduce the remaining bandwidth. The maximum buffer size for n fractions can therefore be expressed as:

$$Max\_buffer = \frac{1}{f_n} R_n - effort$$

From Equation 5.16, it is known that the remaining bandwidth per frame, R_n, can be expressed as:

$$R_n = R - \sum_{i=0}^{n-1} f_i S$$

Substitution provides the following equation for the maximum buffer size:

$$Max\_buffer = \frac{1}{f_n} \left(R - \sum_{i=0}^{n-1} f_i S \right) - effort$$

It is known that the $effort$ is a result of the scheduling of a slot, according to f_n, in the middle of two slots that are scheduled according to f_{n-1}, i.e. it is a result of the start positions of the fractions. The effort to reduce the remaining bandwidth is a result of the collaboration of each individual fraction and can therefore be considered as a series of efforts, a single effort for each fraction. Since the starting position of each of the fractions is located in the middle of the interval determined by the previous fraction, each partial effort can be considered as a function of the previous fraction, the slot size and the remaining bandwidth per frame of the previous fraction:

$$effort_i = 2 \left(\frac{1}{2} \frac{1}{f_{i-1}} R_{i-1} - S \right)$$

Therefore the sum of all partial efforts is formulated as:

$$\begin{aligned}
effort = {} & 2\left(\frac{1}{2}\frac{1}{f_0}R_0 - S\right) \\
& + 2\left(\frac{1}{2}\frac{1}{f_1}R_1 - S\right) \\
& \ldots \\
& + 2\left(\frac{1}{2}\frac{1}{f_{n-2}}R_{n-2} - S\right) \\
& + 2\left(\frac{1}{2}\frac{1}{f_{n-1}}R_{n-1} - S\right)
\end{aligned}$$

By using the equation of the remaining bandwidth per frame, Equation 5.16, the effort can be presented as:

$$\begin{aligned}
effort = {} & 2\left(\frac{1}{2}\frac{1}{f_0}R - S\right) \\
& + 2\left(\frac{1}{2}\frac{1}{f_1}\left(R - f_0S\right) - S\right) \\
& \ldots \\
& + 2\left(\frac{1}{2}\frac{1}{f_{n-2}}\left(R - \sum_{i=0}^{n-3} f_iS\right) - S\right) \\
& + 2\left(\frac{1}{2}\frac{1}{f_{n-1}}\left(R - \sum_{i=0}^{n-2} f_iS\right) - S\right)
\end{aligned}$$

Substitution of the $effort$ in the equation for the maximum buffer size for n fractions leads to:

$$\begin{aligned}
Max\_buffer = {} & \frac{1}{f_n}\left(R - \sum_{i=0}^{n-1} f_iS\right) \\
& - 2\left(\frac{1}{2}\frac{1}{f_0}R - S\right) \\
& - 2\left(\frac{1}{2}\frac{1}{f_1}\left(R - f_0S\right) - S\right) \\
& \ldots \\
& - 2\left(\frac{1}{2}\frac{1}{f_{n-2}}\left(R - \sum_{i=0}^{n-3} f_iS\right) - S\right) \\
& - 2\left(\frac{1}{2}\frac{1}{f_{n-1}}\left(R - \sum_{i=0}^{n-2} f_iS\right) - S\right)
\end{aligned}$$

Simplified, this results to:

$$\begin{aligned}
Max\_buffer = {} & \frac{1}{f_n}\left(R - \sum_{i=0}^{n-1} f_i S\right) \\
& + 2S - \frac{1}{f_0}R \\
& + 2S - \frac{1}{f_1}\left(R - f_0 S\right) \\
& \dots \\
& + 2S - \frac{1}{f_{n-1}}\left(R - \sum_{i=0}^{n-2} f_i S\right)
\end{aligned} \tag{5.34}$$

Note that the equation which needs to be proven, Equation 5.28, can be expanded as follows for n fractions:

$$Max\_buffer = R\frac{1}{f_0} + \sum_{i=1}^{n}\left(\frac{f_{i-1}}{f_i} - 2\right)\frac{1}{f_{i-1}}\left(R - \sum_{j=0}^{i-1} f_j S\right)$$

$$\Updownarrow$$

$$\begin{aligned}
Max\_buffer = {} & R\frac{1}{f_0} \\
& + \left(\frac{f_0}{f_1} - 2\right)\frac{1}{f_0}\left(R - f_0 S\right) \\
& + \left(\frac{f_1}{f_2} - 2\right)\frac{1}{f_1}\left(R - f_0 S - f_1 S\right) \\
& + \dots \\
& + \left(\frac{f_{n-1}}{f_n} - 2\right)\frac{1}{f_{n-1}}\left(R - f_0 S - f_1 S - \ldots - f_{n-1} S\right)
\end{aligned}$$

$$\Updownarrow$$

$$\begin{aligned}
Max\_buffer = {} & R\frac{1}{f_0} \\
& + \frac{1}{f_1}\left(R - f_0S\right) - 2\frac{1}{f_0}\left(R - f_0S\right) \\
& + \frac{1}{f_2}\left(R - f_0S - f_1S\right) - 2\frac{1}{f_1}\left(R - f_0S - f_1S\right) \\
& + \dots \\
& + \frac{1}{f_n}\left(R - \sum_{i=0}^{n-1} f_iS\right) - 2\frac{1}{f_{n-1}}\left(R - \sum_{i=0}^{n-1} f_iS\right)
\end{aligned}$$

$$\Updownarrow$$

$$\begin{aligned}
Max\_buffer = {} & R\frac{1}{f_0} - 2\frac{1}{f_0}\left(R - f_0S\right) \\
& + \frac{1}{f_1}\left(R - f_0S\right) - 2\frac{1}{f_1}\left(R - f_0S - f_1S\right) \\
& + \frac{1}{f_2}\left(R - f_0S - f_1S\right) - 2\frac{1}{f_2}\left(R - f_0S - f_1S - f_2S\right) \\
& + \dots \\
& + \frac{1}{f_{n-1}}\left(R - \sum_{i=0}^{n-2} f_iS\right) - 2\frac{1}{f_{n-1}}\left(R - \sum_{i=0}^{n-2} f_iS - f_{n-1}S\right) \\
& + \frac{1}{f_n}\left(R - \sum_{i=0}^{n-1} f_iS\right)
\end{aligned}$$

$$\Updownarrow$$

$$\begin{aligned}
Max\_buffer = {} & 2S - \frac{1}{f_0}R \\
& + 2S - \frac{1}{f_1}\left(R - f_0S\right) \\
& \dots \\
& + 2S - \frac{1}{f_{n-1}}\left(R - \sum_{i=0}^{n-2} f_iS\right) \\
& + \frac{1}{f_n}\left(R - \sum_{i=0}^{n-1} f_iS\right)
\end{aligned}$$

As can be noticed, the deduced maximum buffer size of Equation 5.34 is identical to the derived equation here, that is, the proof has been provided that for k fraction the equation holds.

Q.E.D.

5.6.3 Maximum latency for n-term slot allocations

As already mentioned in Section 5.6.1, the maximum latency is in direct relation to the maximum buffer size. When analyzing the bandwidth request of 95 bytes per second which can be approximated by the series of fractions $\frac{1}{4} + \frac{1}{16} + \frac{1}{128}$, a similar behavior is detected in the increase of latency in Figure 5.15 as in the accumulated data of Figure 5.12. Note that the results are both based on a frame consisting of eight slots, each having a capacity of 300 bytes per second. Since each fraction is only capable of providing part of the requested bandwidth, the latency increases with each scheduled slot according to the first fraction f_0. The slots allocated according to fraction f_1 result in a temporary decrease in the latency, but the combination of both fractions tends to provide insufficient bandwidth, thereby resulting in a steady increase of the latency due to the backlog of data. Only after scheduling the last fraction, the backlog of data is completely resolved and the cycle can start over again.

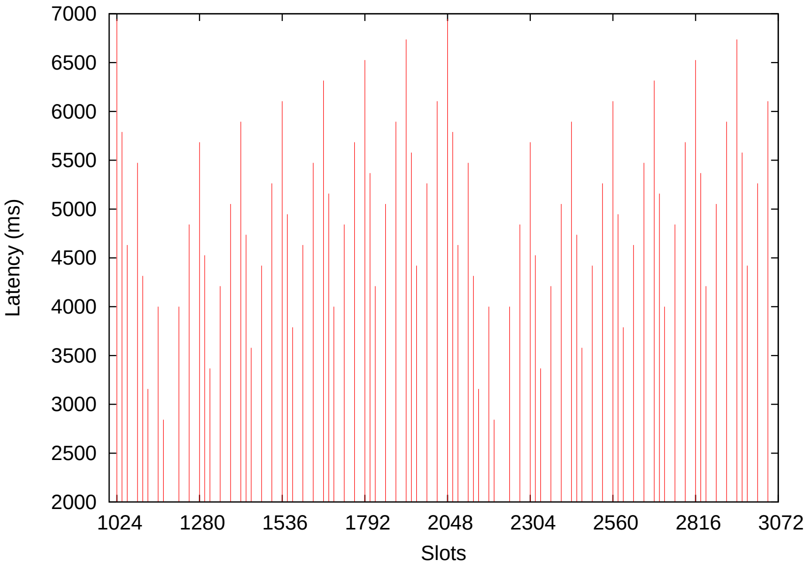


Figure 5.15: *Latency of 95Bps with 300B slots ($\frac{1}{4} + \frac{1}{16} + \frac{1}{128}$)*

Note that while there is a clear correspondence between the buffer size and the latency, there is also a significant difference. The maximum buffer size increases when increasing the requested bandwidth from 77 bytes per second to 95 bytes per second. However, the maximum latency of the 95 bytes per second bandwidth request is lower than that of the 77 bytes per second bandwidth request. Therefore, although the maximum buffer size is larger for a higher requested rate, it could be that the maximum latency is lower. This property is more closely analyzed in this section.

It can be expected that the latency of the small requested amounts will be higher than the larger amounts. Requests that present a lower bandwidth than the capacity of a single slot is scheduled such that the resource is shared with other sensors. The access to the resource is allowed once every x frames in order to provide the sensor to collect a sufficient amount of data to ensure an efficient transmission. While a superficial comparison of the latency of approximations where a clear difference can be observed is relatively easy to accomplish, a more profound comparison of the latency where the approximation exhibits more subtle differences is more difficult.

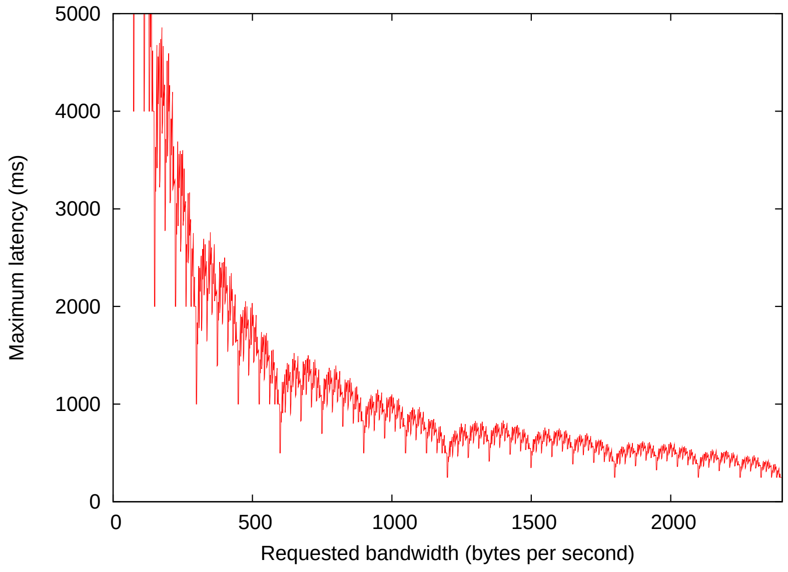


Figure 5.16: *Max latency vs bandwidth request for 300B slots*

A more detailed analysis of the latency is possible by means of interpreting Figure 5.16, which depicts the maximum latency for all the possible integer bandwidth requests constrained by a maximal capacity of 2400 bytes per second. Unlike the behavior of the maximum buffer size, does the maximum latency tend to decrease with an increasing requested bandwidth. The largest maximum latency, 128000 ms or 128 seconds, is observed when requesting a bandwidth of one or two bytes per second, with a frame duration of one second, split up in eight slots, and a slot size of 300 bytes. Due to the relative large slot sizes and the long frame duration, such latency is to be expected. A single slot is scheduled every 128 frames, where a frame has a duration of one second. Note that the algorithm employs a boundary on the precision of the approximation, such that the lowest possible unit fraction has a denominator of 128.

The function which depicts the maximum latency contains a quadratic element, showing a steep descent of the maximum latency for the lower requested bandwidths. As a result, a relative low maximum latency is obtained rather fast for the requested amounts. The lowest maximum latency which can be observed is 250 ms, which is equal to the duration of two slots. Since the highest possible unit fraction within the approximation is equal to 1/2, the highest frequency a slot is scheduled on a regular base is equal to half the number of slots. When taking into account the slot positioning algorithm, according to which the slots are allocated in an interleaving manner, it is clear that the lowest possible maximum latency is equal to two times the slot duration.

Although the maximum latency is in direct relation to the maximum buffer size, their connection is not immediately visible from the figures. A more detailed view, as in Figure 5.17, reveals a similar behavior as the maximum buffer size. The figure is a section (requested bandwidths between 150 and 230 bytes per second) of the previous figure, which depicts the maximum latency with eight slots. The full red lines indicate the maximum latency for that request,and the dotted blue lines is the binary representation of the partial fractions that appear in the approximation of the requested amount. The top blue line is the smallest fraction ($\frac{1}{128}$), the next blue line is the double of the fraction of the previous line and so on, until the bottom blue line, which is fraction $\frac{1}{4}$. Note that the fraction $\frac{1}{2}$ is not shown here, since all the depicted bandwidth requests have this fraction in their approximation and would be depicted as a straight line in the figure, which would not provide significant information.

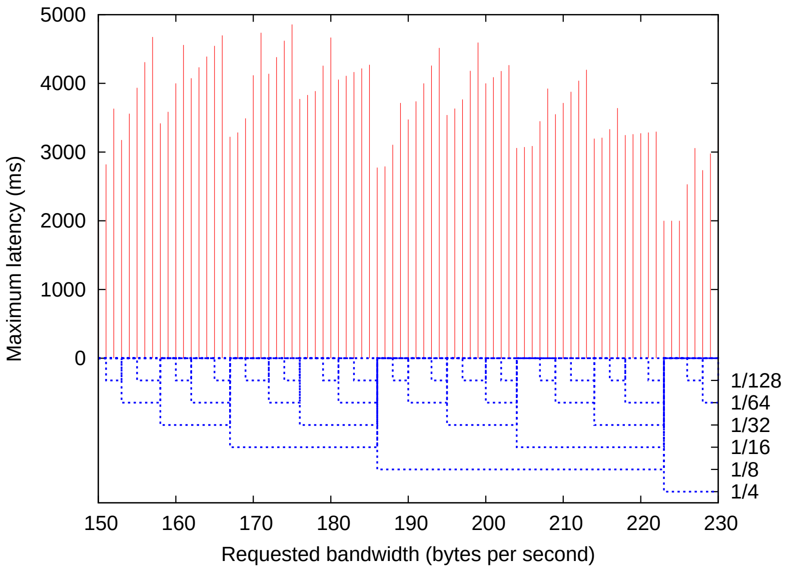


Figure 5.17: *Section of max latency vs bandwidth request for 300B slots*

By examining the composition of the approximations, that is, of which unit fractions the approximation consists, it is clear that even while the maximum latency is descending with larger requested bandwidths, it exhibits increases in the maximum latency when adding another fraction to the approximation, while a decrease is noticed when replacing a series of fraction with only a single fraction. Like with the maximum buffer size, the more fractions are used to approximate the requested bandwidth, the higher the maximum latency might become. However, the maximum latency is proportional to the maximum buffer size and reverse proportional to the requested bandwidth, which is a linearly increasing function. Therefore, when higher bandwidths are requested, the maximum latency might be lower. Small inclinations can be observed when an extra fraction is added to the approximation, however, the requested bandwidth has a large influence on the equation. For example, the top maximum buffer size is at the request of 1999 bytes per second (with 8 slots), while the maximum latency is small at that instance. The high bandwidth request ensures the low maximum latency, since the large buffer is filled at a more rapid pace, thereby resulting in a higher frequency of slots being scheduled to transmit a message. As a consequence, in order to determine the maximum latency, the maximum buffer size can be calculated according to Equation 5.28, from which the maximum latency can be derived by dividing the result by the requested bandwidth.

5.6.4 Discussion

The previous discussions and Equation 5.28 show that the maximum buffer size, and therefore also the maximum latency, is bounded and is therefore predictable. Moreover, the algorithm allows for considerable fine tuning of the maximum buffer size and maximum latency. A case where the waste of bandwidth is to be minimized, but where large latencies are not considered important, could use approximations where the bound of the lowest possible denominator is set very high. On the other hand, in situations where a low latency is required and bandwidth could be sacrificed, the bound on the lowest possible denominator is very low, that is, the approximation becomes more coarse. Besides the accuracy of the approximation, other parameters that influence the maximum latency, such as the slot size, the number of slots, etc. are discussed in this section.

In order to control both the buffer size and the latency, the accuracy of the approximation might be adjusted. The equation and the figures indicate the impact of each of the fractions to the total maximum buffer size. Instead of using a series of fractions to approximate the requested bandwidth, a single fraction, higher than the sum of all partial fractions, can be used to comply to the request by controlling the bound on the highest possible denominator. The reduction in number of fractions leads to a reduction in both buffer size and latency. On the other hand, this results in a higher waste of the available resources, hence, the bandwidth usage efficiency drops. Figure 5.18 illustrates the effect of limiting the number of fractions on the maxima of both the buffer size and latency in otherwise identical circumstances,

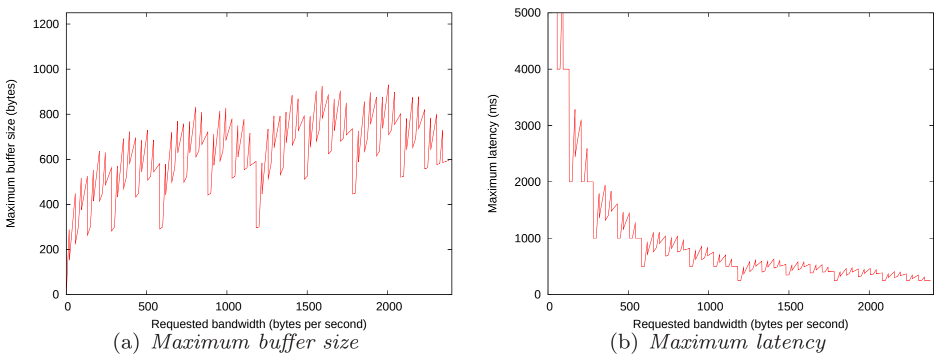


Figure 5.18: *Maxima with denominator limited to* $\frac{1}{16}$

that is, a frame with a capacity of 2400 bytes per second, split up into eight slots. The figures depict the case where the highest denominator is limited to 16 instead of 128, which was used in the previous subsections. It is clear that such limitation successfully controls the maximum buffer size, which was reduced from more than four times the size of a slot to less than three times the size of a slot. Otherwise it can be noticed that the fine granularity of the figure has diminished, resulting in a more coarse view. The improvement on the latency is not immediately noticeable, since the figure depicts only the latency up to 5 seconds. However, one of the most noticeable facts is the reduction of the maximum latency to merely 16 seconds instead of 128 seconds. This is an immediate result of the limitation of the lowest possible denominator. In general, lower latencies are observed, although it is not explicitly clear from the figure since the requested bandwidth reduces the impact of the reduction. The lowest maximum latency has not changed, however, the number of requested bandwidth that result in the lowest maximum latency has. The limitation of the denominator ensures that more requested bandwidths result in the lowest maximum latency.

Besides manipulating the number of fractions that an approximation consists of, the slot size might be adjusted to adapt to the network circumstances. Large slots have the advantage of efficient usage of the wireless medium, since each transmission of data incurs some overhead in the preparation of the radio, and in the transmission of a preamble and PHY header. When the network is designed for high bandwidth sensors, the protocol still enables the low bandwidth sensors to send their data in an efficient manner without wasting too much bandwidth. However, if all sensors in the network only need to transmit a single byte per second, such large slots result in a long buffering time. If the long latency for such small bandwidth requests poses an issue, the number of slots per frame can be increased while maintaining the frame duration. As a result, the size of the slots will be lowered.

By using a frame that is split into 16 slots, means that the slot capacity is halved compared to the slot capacity used in the previous subsections. The capacity of a single slot has therefore become 150 bytes per second. Figure 5.19 depicts the

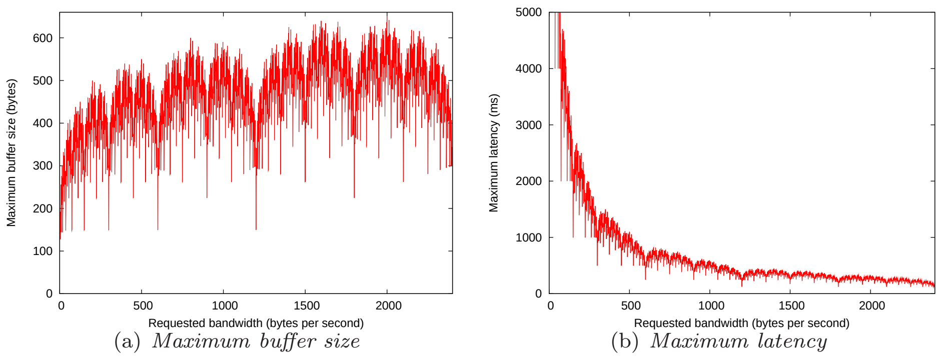


Figure 5.19: *Maxima vs bandwidth request for 150B slots*

maximum for both buffer size and latency as a function of the requested bandwidth for a frame consisting of 16 slots. Note that the gradient and the accuracy of the approximation is maintained while the maximum buffer size has lowered. The requested bandwidths that required previously a shared slot which is scheduled every two frames, are now satisfied with one full slot that is scheduled every frame. The reduction in slot size leads to a decrease in the amount of data that needs to be buffered, thereby shrinking the maximum buffer size. Although the absolute maximum buffer size is reduced, note that the number of buffers, that can handle the capacity of a single slot, has been slightly elevated. For a frame consisting of 16 slots, the absolute maximum buffer size is 650 bytes, thus 4.33 buffers of 150 bytes are needed to host this data. Like the buffer size, the latency is also positively influenced by the smaller slot sizes. Where a sensor needed to collect data during two frames before being allowed to transmit the gathered data, it can now send every frame its collected data. As a result, the latency is also much lower when compared to a slot size of 300 bytes.

5.7 Practical Scheduling Analysis

The previous section considered an analysis of the performance of the proposed scheduling protocol from a theoretical viewpoint for both latency and buffer size. The analysis regards the arrival time of the sensor measurement data as a linear function in time, whereas in reality several bytes of data might be received at the same time. The hardware performing the measurements could buffer a series of measurements and transfer the set of measurements in a burst to the MCU. On the other hand, the sensor might require multiple bytes to represent a single measurement. However, thanks to the abstraction of previous section which introduces some inaccuracies, an equation has been derived to ascertain the maximum buffer size for a specific bandwidth request, which also enables the calculation of the maximum latency. This section places these findings in the right perspective by inspecting the influence of a data arrival pattern in bursts for both the latency and the buffer size. The section considers, like the previous sections, a frame capacity of 2400 bytes per second, with a frame duration of one second. The frame is split into eight slots, each having a capacity of 300 bytes per second. In order to verify the influence of data arriving in bursts, the reception is simulated for a range of one byte to 2400 byte bursts in order to verify the influence of the bursts on the buffer size, discussed in the first subsection, and the influence on the latency, which is considered in the second subsection.

5.7.1 Buffer size

Instead of considering the function for the data arrival as a linear function, the data arrival is regarded as a discrete event, where possibly multiple bytes of data arrive at the same time. Note that the collection of sensor measurements is considered as a periodic event, where each time an equal amount of data is generated. The data arrival can therefore be represented by a step function. When the step size of the data arrival is infinitesimally small, the function representing the arrival approaches a linear function, thereby showing no effect on the previously derived equation for maximum buffer size and latency. As the step size increases, that is, the number of bytes per burst increases, the delicate balance between the arriving data and the transmitted data, which ensures the predictable maximum latency, might become disturbed. Where the theoretical analysis considered a certain amount of data to be available based on the linear function, it might happen that due to the bursts

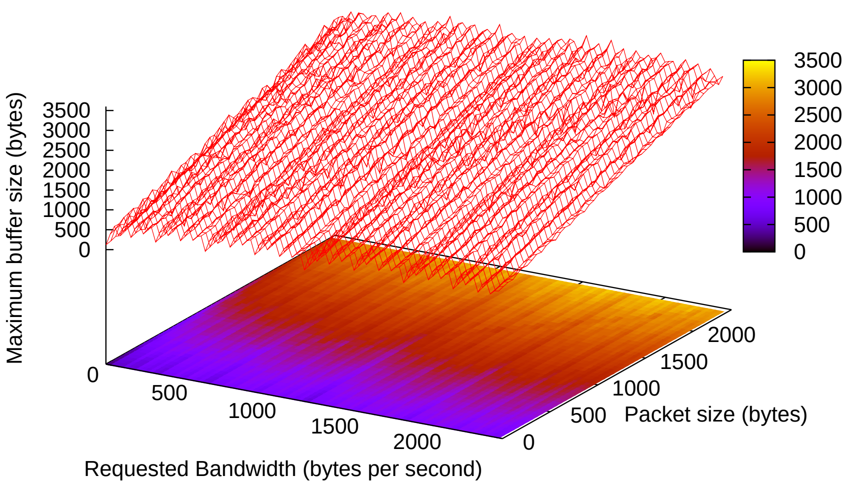


Figure 5.20: *Maximum buffer size vs the requested bandwidth and the packet size*

not all data has arrived yet at the time a new slot is scheduled for transmission. Therefore, the capacity of the slot is not entirely used and more excess data remains in the buffer than expected. In the worst case a slot is scheduled for transmission while no data is available yet.

The influence of the data transmission in bursts on the maximum buffer size is depicted in Figure 5.20 for bandwidth requests ranging from 1 to 2400 byte per second and bursts, also referred to as packet sizes, ranging from 1 to 2400 bytes. Although the figure only depicts two percent of the available information, necessary to make the figure readable, it is clear that the larger the packet size, the higher the maximum buffer size becomes. A closer inspection of the data shows certain combinations of packet size and bandwidth request that produce a lower maximum buffer size compared to other combinations. The patterns produced by these combinations is more clearly depicted in Figure 5.21, where the maximum buffer size is shown as a function of both the packet size and the requested bandwidth. The value of the maximum buffer size is marked with a color code, where a darker pixel represents a lower maximum buffer size. It can be noticed that for the same packet size, a higher requested bandwidth requires a somewhat higher maximum buffer size, however this effect is limited. A more important observation pertains to the lower maximum buffer size along the path of certain vertical, horizontal and diagonal lines.

The vertical minima correspond to the maximum buffer size decrease thanks to the approximation of the requested bandwidth as a larger fraction instead of a series of small fractions, as discussed in Section 5.6. This shows that even though the packet size influences the buffer size, the number of fractions still matter in the determination of the maximum buffer size. This section does not go into detail about this process, since this topic has been discussed extensively in the previous section.

Whereas the vertical minima are clearly visible as a dark line, the horizontal minima are harder to notice. This implies that the difference in maximum buffer size with its surrounding values is not that large. Moreover, the improvement on the buffer size is not always a fact for every requested bandwidth. The improvement is a result of the relation between the packet size and the slot size. Intuitively it can be expected that a positive influence is experienced when the packet size equals or is a multiple of the slot size, since this minimizes the irregularity of the excess amount of data. The simulations in this section employ a slot size equal to 300 bytes and every 300 bytes increase in packet size such horizontal minimum is noticeable.

Both the vertical minima and the horizontal minima are depicted in Figure 5.22, where the maximum buffer size is shown as a function of the requested bandwidth for two different packet sizes, 1190 bytes and 1200 bytes. The vertical minima can be easily noticed as they are represented by the lower maximum buffer sizes in the figure. One of the horizontal minima is depicted as the maximum buffer size function for a packet size with 1200 bytes. It can be noticed that while the other function has a smaller packet size, its required maximum buffer size is larger for a large part of the bandwidth requests. Note also the similarity between the flow with a packet size of 1190 bytes and the flow which was discussed in the previous section, that is a flow for a packet size of a single byte. The higher maximum buffer size values excluded, both flows show a very similar behavior as a function of the requested bandwidth, for which both use the same approximation. The flow of the maximum buffer size for a packet size of 1200 bytes on the other hand shows quantization effects from the bursts. The effect is caused by the packet size which is a multiple of the slot size. A similar effect is present at packet sizes for which the slot size forms a multiple.

The relation between the packet size and the slot size can be examined in further detail in Figure 5.23, which depicts the maximum buffer size as a function of the packet size for a fixed bandwidth request equal to 9 bytes per second. It is clear that the previously established observation about the maximum buffer size increasing with an increasing packet size is confirmed. Interesting to note is the repetitive character of the maximum buffer size according to the packet size. Thanks to the relation between the packet size and the slot size such cycle is possible. Note that the requested bandwidth of 9 bytes per second is approximated by a single fraction, 1/32, and as such the maximum buffer size is equal to 288 bytes according to Equation 5.28. Otherwise formulated, this means that every 32 seconds a slot for transmission is scheduled, which has a capacity equal to 300 bytes. Therefore, when a packet size

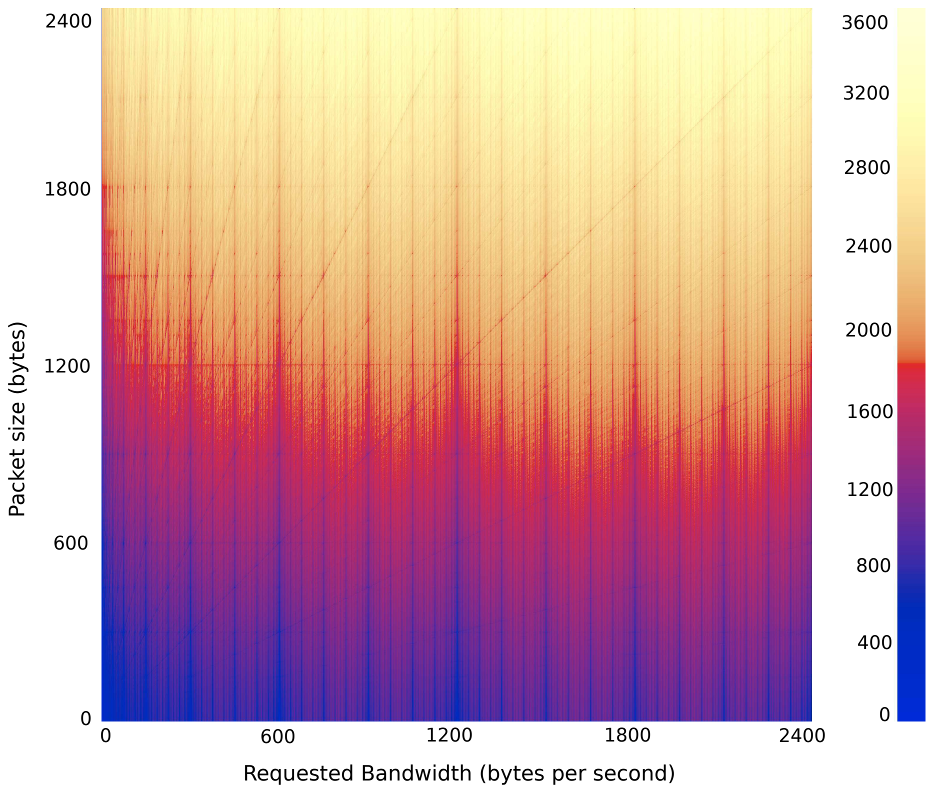


Figure 5.21: *Maximum buffer size for each rate versus the packet size*

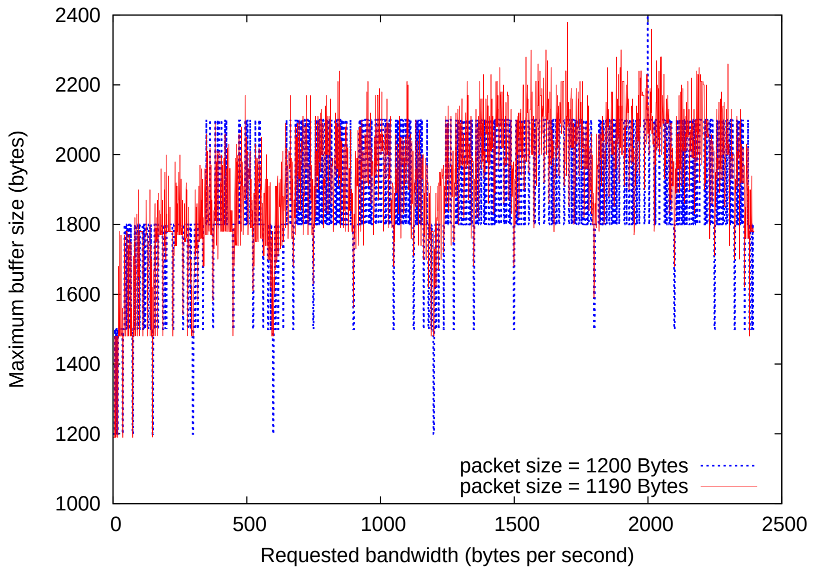


Figure 5.22: *Max buffer size vs the bandwidth, bursts of 1190B and 1200B*

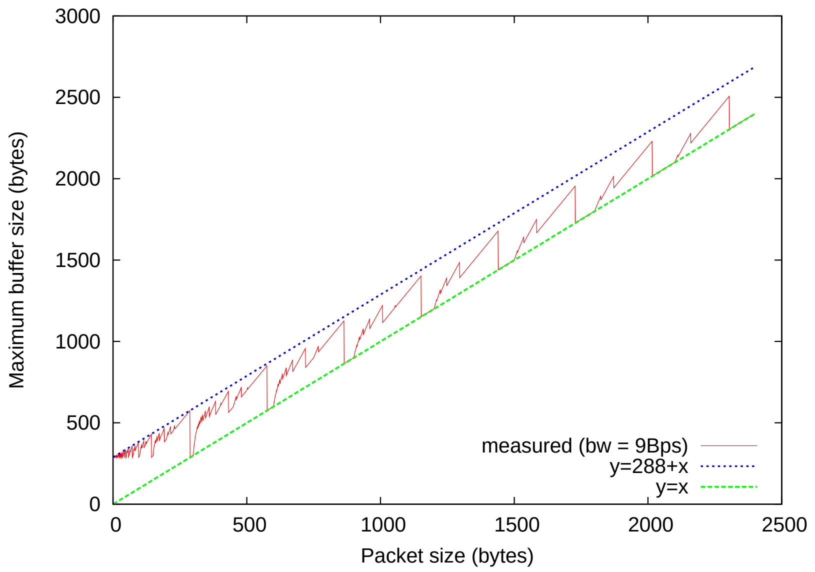
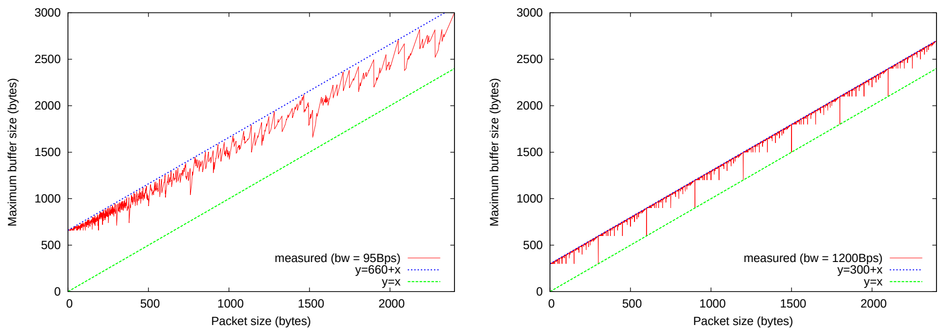


Figure 5.23: *Max buffer size versus the packet size, 9Bps bandwidth request*

which equals 288 bytes is used, the maximum buffer size is equal to the packet size. Note that the packet arrival time is determined both by the requested bandwidth and the packet size. A packet with a size of 288 bytes will therefore arrive every 32 seconds. A packet with the size of 289 bytes will arrive just too late for the first transmission opportunity. The received data, equal to the packet size is stored in the transmission buffer until the next transmission opportunity, which is still before the next packet arrival. As a result a super-cycle is formed of cycles, where packet arrival shifts with respect from the scheduled slot time, until the two overlap again. Since each cycle is capable of transmitting the accumulated data, the maximum buffer size is equal to the packet size. Evidently, each packet size larger than 288 bytes and smaller than or equal to the slot size fulfills this condition. A packet size which is lower than 288 bytes results in a packet arrival which is too soon with an insufficient amount of data. As a result, the packet arrival shifts in the backward position with respect to the scheduled slot times, eventually resulting in the reception of a packet twice during a cycle even before a transmission could have been scheduled. Therefore, the maximum buffer size is increased in this occasion. Note that slot sizes of which a multiple can be formed which is between 288 and 300 bytes, obtain the same maximum bandwidth as their multiple.

Such pattern can be recognized in each of the packet size cycles, which are defined by the slot size. For each of the packet sizes that are a multiple of the slot size, the maximum buffer size is equal to the packet size. In a similar manner as the mechanism for packet sizes lower than the slot size, the maximum buffer size is determined by the packet size. Instead of presenting an equation which defines the exact maximum buffer size, a lower and upper bound for the maximum buffer size is provided, which are depicted in the figure as $f_u(x) = x + 288$ and $f_l(x) = x$. The maximum buffer size determined by Equation 5.28, called theoretical buffer size, also represents the amount of data which is expected to be transmitted at the scheduled slot. Therefore, it also determines whether or not a packet arrives multiple times during a single cycle, that is, whether a larger buffer size is required. As a result, the maximum buffer size can not be higher than the theoretical buffer size increased with the packet size, as demonstrated by function $f_u(x)$. As a result of the bursts, it is also clear that the maximum buffer size can not be lower than the packet size, which is shown by function $f_l(x)$.

Note that these upper and lower boundaries also hold for both higher bandwidth requests and approximations with more fractions. Both cases are depicted in Figure 5.24, where the bandwidth request of 95 bytes per second is approximated as $\frac{1}{4} + \frac{1}{16} + \frac{1}{128}$ and the bandwidth request of 1200 bytes is approximated as 4, indicating an allocation of four slots per frame. Due to the multiple fractions in the approximation of a bandwidth request of 95 bytes per second, the lowest possible maximum buffer size does not equal the packet size, which is clearly not the case with the bandwidth request of 1200 bytes per second. Moreover, the multiple fractions make the recognition of the pattern in the maximum buffer size as a function of the packet size harder, although for the lower packet size it is still feasible.



(a) *Bandwidth request of 95 bytes per second* (b) *Bandwidth request of 1200 bytes per second*

Figure 5.24: *Maximum buffer size versus the packet size*

The previous discussion considered the influence of the packet size on the maximum buffer size for both a fixed packet size and a fixed bandwidth request. However, as could be noticed in the 3D plot, certain combinations of the bandwidth request and packet size ensure a lower maximum buffer size than others. The source for these anomalies needs to be found in the cyclic character of the protocol. In Section 5.6 it was found that the buffer size increases and decreases during a single cycle, of which the duration was determined by the lowest unit fraction contained within the approximation. Due to the data arriving in bursts, a super-cycle is created, where the increase and decrease of the buffer spans several cycles. As already could have been noticed in the previous analysis, when the cycle duration is divisible by the arrival time of the packets, a lower maximum buffer size is achieved. In such case, the super-cycle duration is equal to the cycle duration. This observation can be made in Figure 5.25, where the buffer size is depicted in function of the allocated slots, which are proportional to time. The figure depicts a requested bandwidth equal to 70 bytes per second, which can be approximated as $\frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64}$, and a data arrival in bursts of 280 bytes. Note that although the buffer is filled immediately to its maximum and then decreases with each scheduled transmission, the cycle duration remains the same as with the theoretical linear arrival of the data.

Interesting is the predictability of the duration of the super-cycle, for which an equation can be derived. In order to determine the number of cycles a super-cycle is composed of, the common factors between the packet size P, requested bandwidth R and the duration of the cycle D need to be determined. A popular common factor which allows to specify a measure of divisibility is the greatest common divisor (gcd). Between each of the parameters, the gcd can be easily determined and can be used to express their relationship. The packet size can therefore be expressed as follows:

$$P = x \gcd(R, P)$$

Note that the arrival time of the packets can be expressed in terms of the packet

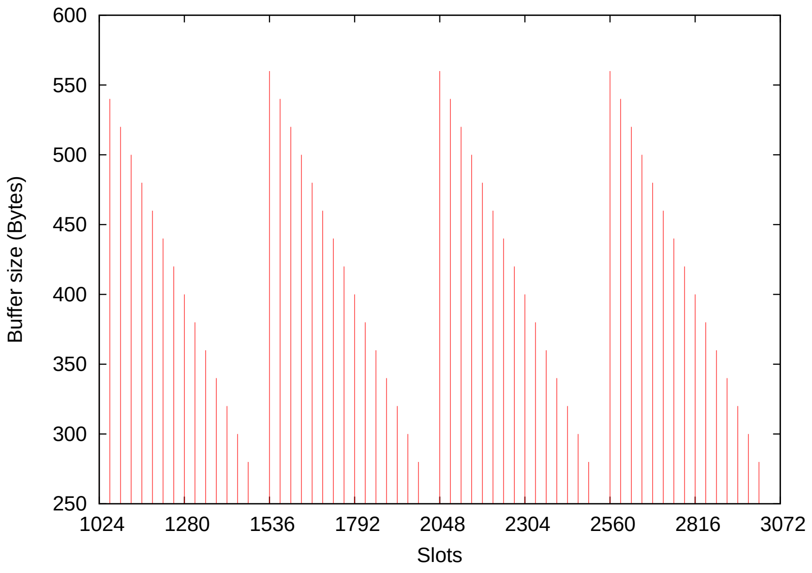


Figure 5.25: *Buffer size of 70Bps with 300B slots and bursts of 280B*

size and the requested bandwidth as:

$$arr\_time = \frac{P}{R} = \frac{x\,\mathrm{gcd}(R,P)}{R}$$

The relation of the cycle duration and the packet arrival time is the topic of interest, which can provide valuable information. If the cycle duration is divisible by the arrival time, then the super-cycle consists of only a single cycle, which can be expressed by:

$$modulo(D, arr\_time) = 0$$

A different notation to represent the demand that the cycle is divisible by the arrival time is:

$$D \,\big|\, arr\_time$$

By means of substitution the following equation is obtained:

$$DR \,\Big|\, x\,\mathrm{gcd}\big(R,P\big)$$

From this equation, it is known that R is divisible by $gcd(R,P)$ and therefore DR is divisible by $xgcd(R,P)$ if DR is divisible by x. The relation between x and D can be expressed as follows:

$$x = y\,\mathrm{gcd}(D,x)$$

Therefore, the demand that the cycle is divisible by the arrival time can be expressed as:

$$DR \,\big|\, y\,\mathrm{gcd}\big(D,x\big)\,\mathrm{gcd}\big(R,P\big)$$

As a result, it can be said that the scheduling period is divisible by the arrival time if y equals to one. Even more, y represents the number of cycles the super-cycle consists of. It is known that P and x can be represented respectively as:

$$P = y * \gcd(D, x) * \gcd(R, P) \qquad \textit{and} \qquad x = \frac{P}{\gcd(R, P)}$$

Therefore, in order to determine the value of y, the following equation can be derived:

$$y = \frac{1}{\gcd\left(D, \dfrac{P}{\gcd(R, P)}\right)} * \frac{P}{\gcd(R, P)} \tag{5.35}$$

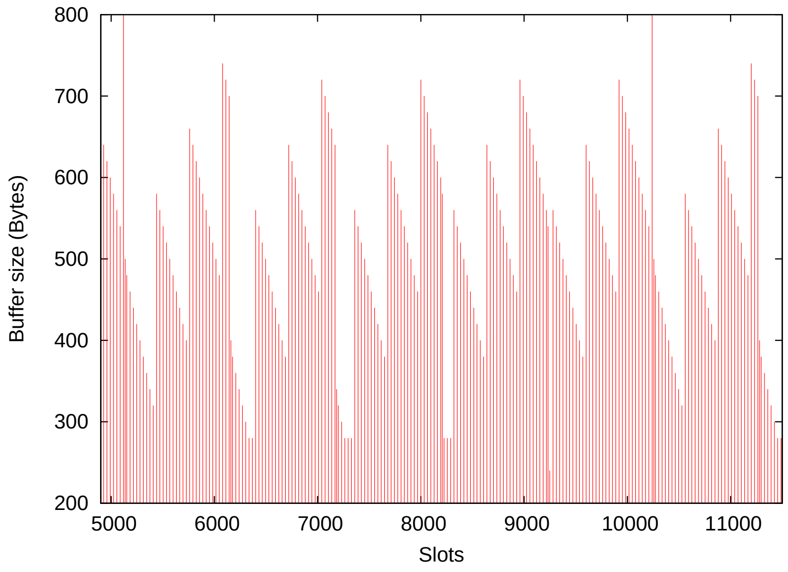


Figure 5.26: *Buffer size of 77Bps with 300B slots and bursts of 280B*

As can be noticed, the number of cycles contained within a super-cycle depends on the packet size, the requested bandwidth and the duration of the cycle. As a result, the combinations of the requested bandwidth and the packet size that result in a super-cycle containing only a single cycle, and thereby a lower maximum buffer size, are discerned as the diagonal minima in the 3D plot. In order to show the effect of the number of cycles within a super-cycle on the buffer size, it is depicted in Figures 5.25 and 5.26 for requested bandwidths respectively equal to 70 and 77 bytes per second and a packet size of 280 bytes for both simulations. The bandwidth request of 70 bytes is approximated as the series of fractions $\frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64}$, while the request of 77 bytes per second is approximated as $\frac{1}{4} + \frac{1}{128}$. The approximations for a requested bandwidth of 70 bytes per seconds and 77 bytes per second have a cycle duration of

64 seconds and 128 seconds respectively. According to Equation 5.35, the number of cycles within the super-cycle for the requested bandwidth of 70 bytes per second is equal to one, since $gcd(R, P) = gcd(70, 280) = 70$, $P/gcd(R, P) = 280/70 = 4$ and $gcd(D, P/gcd(R, P)) = gcd(64, 4) = 4$, resulting therefore in the number of cycles within the super-cycle equal to $4/4 = 1$. This is evident from Figure 5.25, where all 280 bytes are transmitted within a single cycle. On the other hand, for the requested buffer size of 77 bytes per second, the number of cycles within a super-cycle is equal to five. The $gcd(R, P) = gcd(77, 280) = 7$, $P/gcd(R, P) = 280/7 = 40$ and $gcd(D, P/gcd(R, P)) = gcd(128, 40) = 8$, thereby resulting in a number of cycles equal to $40/8 = 5$. In Figure 5.26 it is clear that while the cycle duration is 1024 slots, determined by the lowest unit fraction, the super-cycle has a duration of 5120 slots, which is five times the cycle duration. It can also be noticed that due to the increased duration of the periodic cycle, the maximum buffer size increases. The arrival timings of the data does not match with the scheduled transmission slots according to the unit fractions of the approximations. As a result, the packet arrival might occur two times during one of the cycles, generating an excess of data, which otherwise would have been transmitted in a regular pace.

5.7.2 Latency

The previous subsection showed an extensive analysis of the maximum buffer size in an environment where the data arrives in bursts. Because of the direct relation between the maximum buffer size and the maximum latency, which was already established in Section 5.6, the analysis of the influence of the packet size on the maximum latency remains limited. The analysis in this subsection is also performed with a frame capacity of 2400 bytes per second, eight slots per frame and packet sizes ranging from one byte to 2400 bytes. An all encompassing result can be seen in Figure 5.27, which depicts the logarithmic function of the maximum latency over the requested bandwidth and the packet size. The maximum latency is represented in a logarithmic scale in order to highlight the details of the upper bandwidth requests. Note that the figure includes only 2% of the available information in order to increase the readability.

It can be observed that irrespective of the packet size, a lower bandwidth request results in a higher maximum latency due to the long duration of the cycle. Moreover, as with the maximum buffer size, does an increasing packet size negatively influence the maximum latency. However, the impact of the packet size is more pronounced in the maximum latency for lower bandwidth requests. The same packet size results in a larger increase for low bandwidth requests. Besides the most obvious observations, it can also be noticed that the maximum latency exhibits similar paths representing minima in the horizontal, vertical and diagonal direction, as with the maximum buffer size.

The vertical minima are a result of the reduction of the number of fractions used to approximate the requested bandwidth. The maximum latency in function of the re-

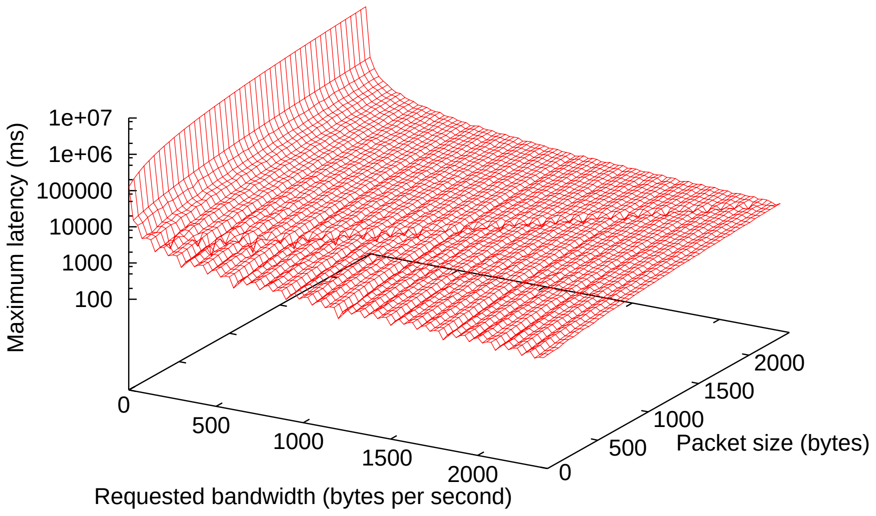


Figure 5.27: *Maximum latency vs the requested bandwidth and the packet size*

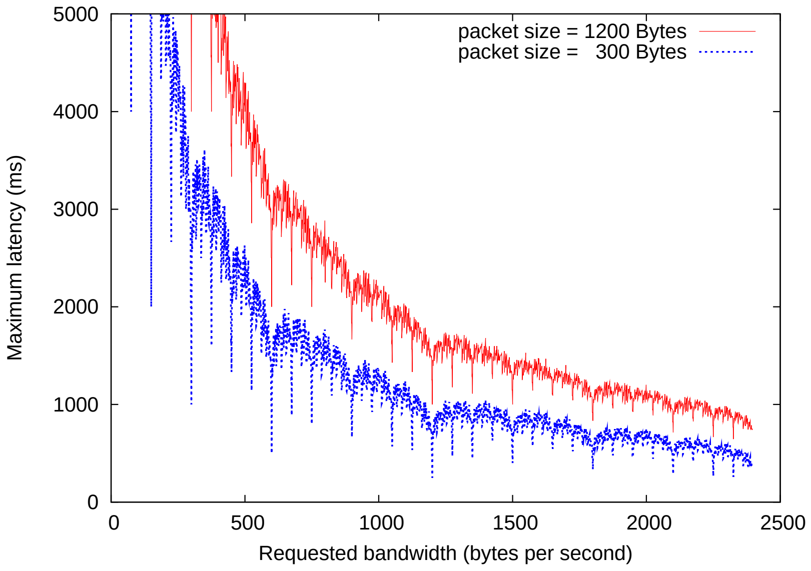


Figure 5.28: *Max latency vs the requested bandwidth, bursts of 300B and 1200B*

quested bandwidth is depicted in Figure 5.28, where the maximum latency is shown for two fixed packet sizes, 300 bytes and 1200 bytes. Although the packet size influences the maximum latency, the composition of the approximation is controlling the relative maximum latency values. Note also that the packet size is having a larger impact on the maximum latency for lower bandwidth requests. The maximum latency can be derived from the maximum buffer size by means of a division by the bandwidth request. Since the maximum buffer size does not experience a large difference between low bandwidth requests and high bandwidth requests, irrespectively from the packet size, the maximum latency for higher bandwidth requests experiences a lower increase due to the packet size.

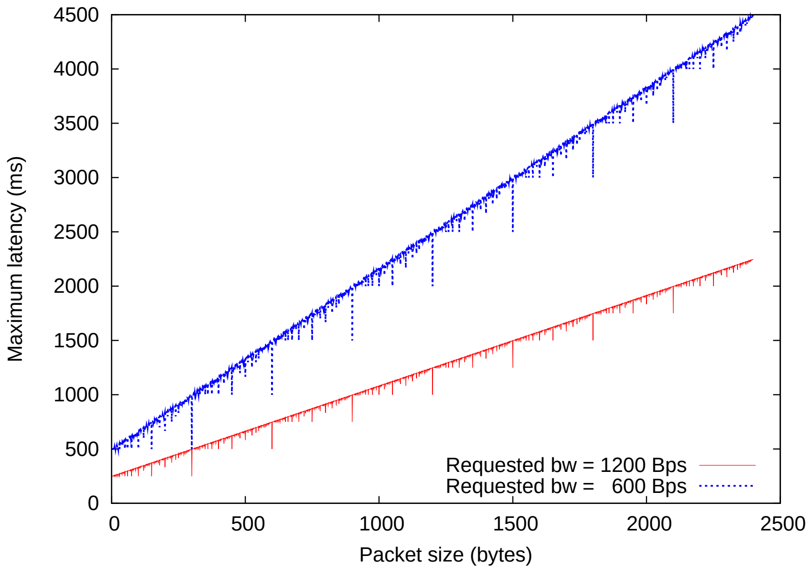


Figure 5.29: *Maximum latency vs the packet size, 600Bps and 1200Bps bandwidth*

The horizontal paths that represent packet sizes for which the maximum latency is lower than in its surrounding packet sizes are a consequence of the divisibility of the packet size by the slot size. The minima are shown clearly in Figure 5.29, which depicts the maximum latency in function of the packet size for two different bandwidth requests, 600 bytes per second and 1200 bytes per second. The minima are visible when the packet size is a multiple of the slot size. Since the arrival time of the packet size matches with the scheduled slot time according to the theoretical linear arrival function, no excess data is generated and therefore also no extra delays. A similar reasoning can be followed when considering packet sizes for which the slot size is a multiple, which is also clearly depicted in the figure. Note that the maximum latency of the different bandwidth requests follow a different gradient, indicating the influence of the bandwidth request in the impact of the packet size.

The diagonal minima, are like in the maximum buffer size case, a consequence of certain combination according to which a minimum is reached. The combinations which ensure a super-cycle consisting of only a single cycle lead to the lowest maximum latencies. The Equation 5.35 can be used to determine these combinations.

5.8 Networked Scheduling Analysis

The previous sections brought an analysis of the performance of the allocated slot schedule according to the protocol for both a theoretical linear data arrival and a data arrival in bursts. However, the analysis remained limited to the specific sensor node, even though the protocol is specifically designed to operate in a networked environment, thereby reducing the amount of overhead, taking into account the lossy character of the wireless medium, etc. By ensuring that nodes need to be informed about their slot allocation only once, the protocol creates an environment where the new nodes do not interfere with the existing schedule and the established schedule does not need to be modified to accomplish the allocation for newer nodes. Moreover, the schedule allows both high throughput sensor nodes and low throughput sensor nodes to coexist in a fair manner in the same network, without affecting the efficiency or network performance. This section investigates the fairness of the protocol between different sensor nodes by simulating a network environment. Note that the analysis of the performance of the scheduling protocol in a network environment remains on a superficial level. A simulation environment is created according to a model representing the real life situation. However, the model often introduces some abstraction, simplification or assumption such that the results do not perfectly match with the reality, although a general idea can be derived from the results. Since the following section discusses the implementation and experimentation of the scheduling protocol on sensor devices, this section is limited to a rudimentary discussion.

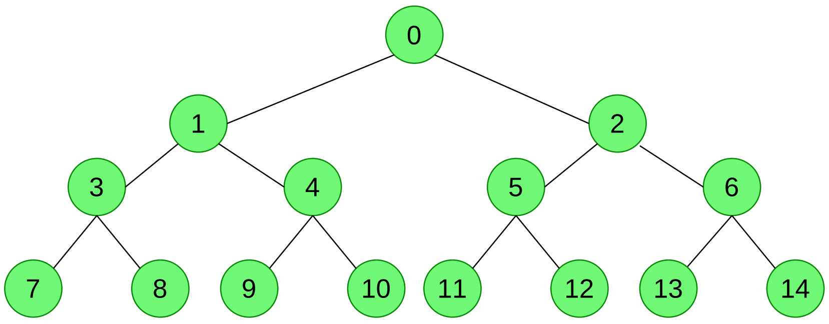


Figure 5.30: *The network topology and node numbering*

The created simulation setup considers a tree based slotted network, such as depicted in Figure 5.30, where the slot allocation is grouped into frames, which are equally sized. In order to approximate the reality as closely as possible, the arrival, and therefore also the bandwidth request, of new nodes is included in the simulation. The joining of a sensor is established by informing its closest neighbor of the requested bandwidth. The neighboring node, which will act as the future parent of the joining node, determines whether it requires more bandwidth, or is capable of supplying the requested bandwidth by the already available resources. Either way, the parent node informs its own parent of the additional bandwidth it will need to transmit, along with any possible additional resource request to supplicate its recently acquired child. All nodes contact their respective parent, until the root node receives the update on the bandwidth usage of its own child. Note that the root is not aware of all individual bandwidth requests, just the required bandwidth its child needs. The root node determines the slot allocation according to the newly requested bandwidth and forwards the information to its child, which allocates half of the slots for itself and half for its child. This process continues until the recently joined sensor node has received a slot allocation schedule. Note that if a parent node is capable of supplying sufficient bandwidth to the child, it determines the slot allocation based on its available resources without the assistance of the root node. Interesting about this protocol is the amount of data that is required to inform the sensors about the slot assignment. The slot index and frame index of the first to be scheduled slot suffices if the periodicity of the slot allocation is known. Therefore, for each fraction, the frame and slot index of the first scheduled slot needs to be provided such that nodes are capable of determining the slot allocation based on the local information.

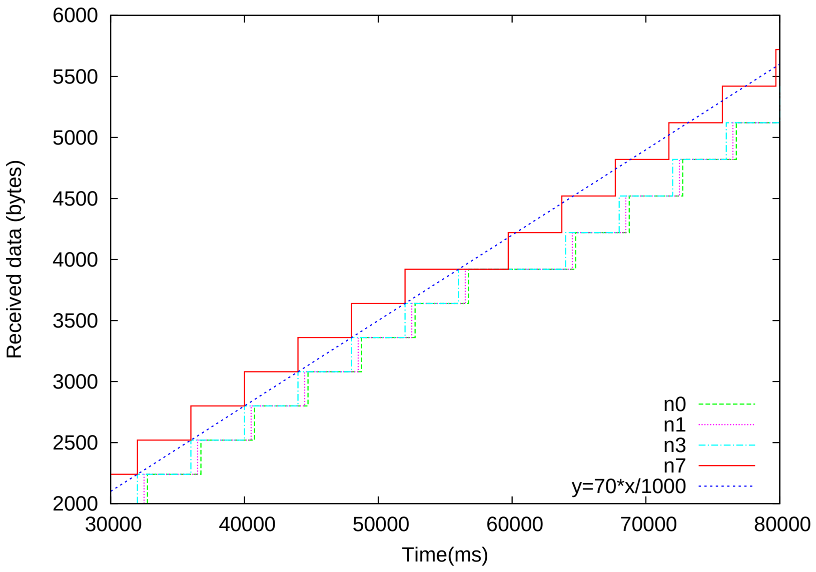
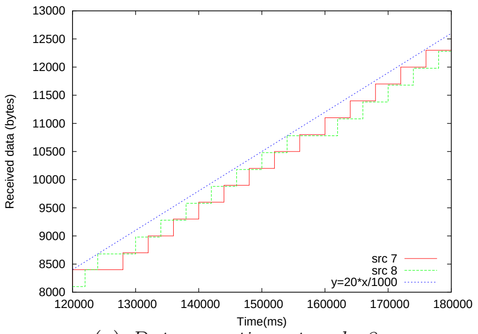


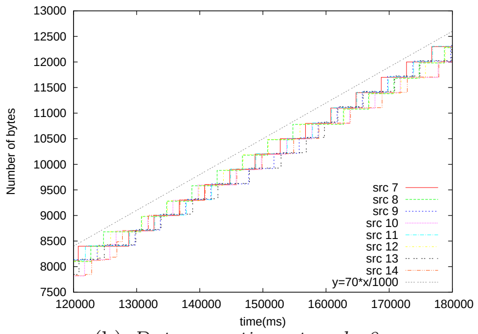
Figure 5.31: *Received data originating from node 7 at a rate of 70Bps*

The simulations in this section assume a physical throughput of maximum 19200 bits per second, that is 2400 bytes per second. A frame is considered to have a duration of one second and is split up into a total of 32 slots, each therefore having a capacity of 75 bytes per second. In the depicted setup, all eight leaf sensor nodes generate data at an equal data rate, which ranges from one byte per second to 75 bytes per second. This section is focused solely on the theoretical performance of the scheduling protocol in a network and therefore assumes that all data from children can be aggregated. The aggregation respects the size of the received data, however, it disregards any possible additional header information which is required to discern the data from the different nodes. The practical issues regarding the aggregation are for the moment ignored. The following section will consider an alternative approach, thereby taking into account the practical limitations.

The analysis of the influence of the network operation on the performance of the protocol consists of the verification of the data reception pattern on the one hand and the study of the latency on the other hand. The data reception pattern of the analysis is depicted in Figure 5.31, which shows the arrival times of the data for all sensor nodes on the path from sensor node 7 ($n7$) to the root node ($n0$) for a data source generating information at a rate of 70 bytes per second. The step events indicate the time at which the first byte is received. Since the node 7 needs to gather the data in a steady pace, the plot shows a certain duration before the packet is received at the parent node. Thanks to the equal distribution of the allocation to slots, also among different levels in the tree, a gradual and reliable reception of the data is ensured, which follows the slot allocation schedule.



(a) *Data reception at node 3*

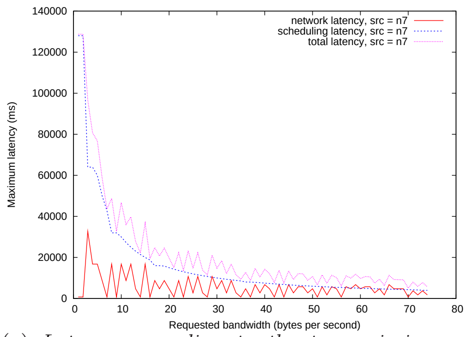


(b) *Data reception at node 0*

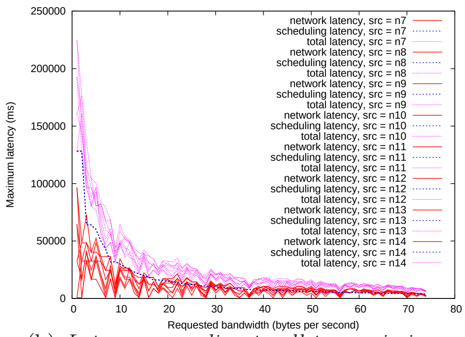
Figure 5.32: *The received data from the perspective of nodes 3 and 0*

Note that the former figure considered only the data which originated from node 7. The Figure 5.32 shows the scheduling of the data at node 3, which receives data from only two children, and the data at the root node, which receives all data. The first subfigure depicts the transmitted data from both node 7 and node 8, for which node 3 acts as parent. It is clear that both sensors transmit their data around the same time with respect to the duration of the cycle. While both nodes are assigned an equal number of slots, the offset of both schedules, that is, the offset of the initial slot, is equal to half the period. As a result, the data from node 7 arrives at first earlier than the data from node 8 and after half the period, this relation is switched, where the data from node 8 is arriving earlier. On average, both nodes are considered to transmit an equal amount of data during a certain time interval. The right subfigure of Figure 5.32 depicts all transmitted data being received at the root node. Note that even though more data is available, the resulting schedule maintains the fairness and when data is at first arriving sooner than other data, the order in which data is received rotates, such that in the end the root has received an equal amount of data from all nodes.

The influence of the network operation on the latency is analyzed in Figure 5.33, where the maximum latency of the data originating from sensor node 7 is depicted in the left figure and the maximum latency from all data on the right figure. A distinction is made in the type of latency based on its cause. The scheduling latency is the maximum latency introduced by the operation of the protocol, previously discussed in 5.6. The network latency is the maximum latency experienced due to the transfer of the data from the originating node to the root node, and the total latency is considered the sum of both latencies. Note that when comparing the actually obtained scheduling latency with the theoretical maximum latency for a frame consisting of 32 slots, a large resemblance can be found. The network latency is a result of subtracting the scheduling latency from the total measured latency. Note that the impact of the network latency decreases with an increasing requested bandwidth, like the scheduling latency. When deriving the network latency from all data flows received at the root node, depicted at the right of the figure, it can be noticed that the network latency for all data flows exhibits a rather similar behavior. Therefore, it can be said that the network latency can vary, but on average, is deemed to follow a similar behavior.



(a) *Latency according to the transmissions of node 7*



(b) *Latency according to all transmissions*

Figure 5.33: *Maximum latency vs the bandwidth request*

5.9 Realistic Scheduling Analysis

The previous sections focused on the description and analysis of the slot allocation protocol, where the analysis consisted of simulations where certain assumptions or abstractions were made in order the maintain the focus on the performance of the protocol, and therefore do not represent an exact model of the reality. As such, the simulated network did not consider any overhead due to the required synchronization, and did not include the overhead of preambles, message headers or switching times. The available bandwidth was entirely used for the transmission of data. This section will discuss not only a realistic implementation of the slot scheduling protocol, but will incorporate it into a complete system, which ensures synchronization, provides methods for joining and leaving nodes, and ensures an autonomous operation, including the self-sufficient construction of the spanning tree.

The entire TDMA protocol is designed for and executed on TelosB sensor nodes [160], which are already extensively discussed in Section 4.5.2 within the context of time synchronization. The implementation is constructed within TinyOS [161], which is a set of modules, comprised of a work scheduler and a collection of drivers for several commonly used sensor platforms. The entire code base is written in nesC, a dialect of C. More information about TinyOS can be found on the Wiki page. This section will not disclose the details of the implementation in TinyOS, since this would be out of the scope of this chapter, while the added value remains questionable. However, a global overview of the implementation and the operation of the protocol on a hardware platform will be provided, especially when the design choices are related to the adjustment of the protocol to the real life conditions.

Note that the previous section assumed aggregation of the data, where the amount of data was respected but no overhead was introduced. In reality, the aggregation of the data leads to some negative effects. A trivial downside of the combination of the data from two sources into a single packet is the required overhead to be able to discern the amount of data and source of the data at reception at the root node. Since the bandwidth request is approximated as precisely as possible in the slot allocation, no or little room remains available to safeguard the operation of the protocol while providing bandwidth for the overhead. The aggregated data and the additional overhead could result in a total amount of data which is too large to be comprised within a single packet, thereby requiring the fragmentation of the packets and thus an additional overhead with regards to the data header. Such operation would lead to higher latencies, thereby impairing the determinism of the slot allocation protocol. Additionally, the overhead might accumulate over the different levels of the tree, by the aggregation of aggregated packets. With each level, the number of bytes that can be effectively used for data is reduced due to the aggregation overhead. A second disadvantage of the aggregation is the transmission time. The scheduling protocol is deterministic, such that a node is aware of the scheduled receptions. Therefore, even before the packet has been received, the sensor node is capable of preparing a transmission which needs to be scheduled in the future. Even if the transmission slot is right after the reception slot, such operation is feasible. On the other hand, when aggregation is being employed, the data needs to be copied in the corresponding transmit buffer, the header needs to be adjusted, etc. As a result, aggregation would diminish the efficiency of the system in terms of latency. Therefore, the implementation relinquishes aggregation and transmits every packet as it was received at the cost of a less efficient bandwidth usage. The data of two children requesting half a slot per frame could be transmitted in a single slot per frame in theory, but due to the previously discussed issues, the parent node requires a slot per frame for each of the children. This design easily leads to a waste of bandwidth when having a tree with a huge number of levels. Therefore, the experimentation in this section only considers a specific type of tree, that is, a star topology, even though the design is not limited to such constraints.

In the remainder of this section first the design of the protocol is discussed, which includes the synchronization, the identification of messages and the autonomous network operation. Afterwards, the performance section elaborates on the experiments and the obtained results.

5.9.1 The design

The TDMA protocol should provide a means to synchronize the nodes, construct the spanning tree such that the most suitable neighbors are selected as parent nodes, ensure the allocation of slots according to the slot allocation algorithm while taking into account the required switching time between TX,RX and idle states, the transmission preamble and the data headers, etc. The control functions of the TDMA protocol is partly based on the functionality which was discussed in 4.8, where the implementation was based on the Magnetic Induction Radio ASIC (MIRA). Whereas the MIRA provided an internal task scheduler and related superframe counter, the TelosB platform does not support these functionalities in hardware. Therefore, a TDMA scheduler module is designed in software alongside a superframe counter in order to support the accurate scheduling of tasks such as a transmission or receive order. Each task is associated with a start time, relative to the superframe counter, a duration and task specific information. Since the precision of the scheduling of the transmission and reception is critical for a reliable operation, the incremental operation of the superframe counter needs to be as precise as possible. Therefore, instead of using the internal clock with a higher resolution, the more stable external 32 kHz crystal is used as clock source, which provides an accuracy that is near to the time required to transmit a single byte. By means of this clock source, only a small synchronization correction is required every four seconds.

The synchronization mechanism used in this implementation was already discussed in 4.8 for the Magnetic Induction Radio ASIC. For the TelosB platform, the synchronization would induce the synchronization of the superframe counters of both nodes. Basically, the transmission opportunity is fragmented in slots, of which a certain number of slots are dedicated to the synchronization. A synchronization message consist of, amongst others, the index of the synchronization slot in which it will be sent. Even though the TelosB platform is composed out of two chips, the microprocessor and the radio chip, the design allows the radio chip to signal the microprocessor of the arrival time of a packet. Upon the detection of the Start of Frame Delimiter (SFD) of the packet, which is added during the transmission of the packet by the hardware, the radio sends an interrupt to the microprocessor. The interrupt triggers a specific functionality of the internal timer module of the microprocessor, such that at the time of the arrival of the SFD, the current value of the timer is stored in a separate register. Since this timer is configured as the superframe counter, a precise time indication is provided for the arrival of the SFD marker. Knowledge about the length of the preamble allows to deduce the start of the slot. Since the offset of the synchronization slot towards the start of the frame is fixed at design time and therefore known by all nodes, the timestamp of the reception of the synchronization message can be used in combination with the synchronization slot index to determine the exact frame start of the transmitter. After the initial synchronization, the synchronization message is used for minor adjustments to the superframe counter to compensate for the drift between the different clocks. The compensation includes a single increment or decrement of the counter such that it does not result in any loss of packets.

In order to support the dynamic joining and leaving of nodes, a second type of slots is defined as *identification slots*. A node attempting to join the network listens for a number of frames to gather statistics about the received synchronization messages. Since the node has no notion regarding the start of the frame yet, it listens for a long duration to the wireless medium until a message is detected. Detected messages which do not comply to the format of a synchronization message are not considered during this stage of the process. The node with the strongest signal is selected to act as the parent of the joining node by synchronizing to the received messages. Only after the sensor has verified its synchronization is successful by listening for a subsequent synchronization message of the same node at the expected time, the sensor is allowed to start sending messages.

During the *identification slots* the joining node announces its presence to the parent node by sending it a message containing the ids from both nodes and the requested bandwidth. Since multiple nodes might attempt to announce themselves at the same time, a simple random backoff mechanism is employed based on the uptime of the sensor. The least significant bits of the uptime are used to determine a backoff time, which can be expected to be different from other nodes. The node receiving such identification message needs to allocate sufficient bandwidth to allow the reception of the data of the joining child. If the node has no or insufficient bandwidth available to support any children, it needs to request extra slots to its own parent, along with an update considering the required bandwidth, since the extra bandwidth of the child is added to the bandwidth already reserved by the parent node. Note that the extra bandwidth of the child is not to be just added to the already consumed bandwidth, since aggregation is only supported for data coming from the same source, not from different sources. As a result, even if the extra bandwidth might be supplied by the already allocated bandwidth by combining the data sources, the node will need to ask for a new slot due to the prohibition of aggregation. The additional requested bandwidth traverses all parent nodes of the respective nodes, each specifying the required bandwidth to service the request, until the request reaches the root node. The root node approximates the bandwidth request as an Egyptian Fraction and selects the slot and frame index which indicate the initial slots for each individual unit fraction comprised within the approximation. Note that the root node has no information about the bandwidth consumption of each individual sensor in the network, it only knows the required bandwidth for each of its children. The newly allocated slot information is included in the following synchronization message that is to be scheduled. The node to which the allocation information is directed updates its available bandwidth information and distributes according to the same principle as the root node the requested bandwidth to its own children. The difference is that the root node has an unconstrained repository of slots within the boundaries of the frame duration and available bandwidth. The regular parent nodes only have a limited set of slots to their disposal to allocate bandwidth to their children. The process of slot distribution occurs until the joining node is provided with sufficient slots to supplicate its requested bandwidth, after which it can start sending data in the slots dedicated for data transmissions, according to the received transmission schedule. The association procedure requires a number of cycles, depending on its depth within the tree and whether or not the transmission slot of the parent is located after the reception slot. During tests, it was observed that the synchronization and association phase of a shallow tree took no more than 3 cycles before new nodes could start transmitting their data. However, there are a lot of parameters that could influence this behavior, such as the interference to noise ratio, the number of hops a node requires to reach the root node, the position of the association slots, etc. The design focused deliberately on a possibly slower association procedure, while still not being inefficient, for the benefit of stability of the network in general.

Note that during the process of the slot allocation, the joining node might have retransmitted its bandwidth request to its parent. This message leads to retransmission of the requests until a node is reached which has been already assigned the required slots and will respond with either sending the slot allocation for the first time, or a retransmission of the slot allocation. The repetition of the bandwidth request will therefore not result in a bandwidth allocation of twice the requested bandwidth.

While the bandwidth request procedure might seem elaborate, it should be noted that the entire bandwidth request and allocation method needs to be performed only once, before the actual data transmission of the sensor even starts. Moreover, the procedure does not interfere with the already scheduled slots. The control messages are passed to parent or child though dedicated control messages and the newly created schedule does not interfere with the existing schedule thanks to the slot allocation algorithm.

While the implementation could have used the regular message passing mechanism of TinyOS, a different approach was followed in favor of the efficiency. The original message passing forces numerous memory copy operations and defines a fixed message size, which needs to be designed such that it can accommodate the largest possible packet. As such, the approach results in an inefficient usage of both memory and CPU resources. As an alternative, this implementation makes use of a memory heap in which a series of circular buffers can be defined. Each buffer is identified according to its index and is associated with parameters that indicate the specific number of buffers and size of each buffer. The TDMA scheduler allocates a specific buffer of the right type to a scheduled task, which can be used to either transfer data from or to the buffer, depending on whether the task involves a transmit or receive operation. The ring buffer method ensures that several tasks with pointers to different buffers can be placed in the task queue without resulting in memory corruption. The processing of the received packets can therefore also happen in the background and does not need to be processed immediately to free the used memory. Moreover, since there is an inverse relation between the number of data slots and the size of the data slots, the slot allocation schedule is also contained within the heap containing the ring buffers. As the number of slots increases, the slot allocation schedule increases and the ring buffer of the data slots decreases. Therefore, both memory constructions provide an equilibrium in the amount of required memory. As a result of the ring buffer, no unnecessary memory copy operations are required, nor is the memory wasted to unused memory allocations.

5.9.2 Performance evaluation

The performance analysis is executed on a network of TelosB sensor nodes, forced into a star topology by only allowing the root node to broadcast synchronization packets. In order to make a well-founded comparison with the simulation section, the data is generated at a pace of one byte at a time at the sensor side according to the requested bandwidth. In order to eliminate possible rounding errors by performing floating point calculations on an embedded device, the requested bandwidth is depicted as the number of bytes per frame, instead of byte per second. The conversion needs to be performed at compile time. The experiment does not consider the entire range of possible bandwidth within the boundaries of the available bandwidth, only a set of bandwidths that are interesting are selected. While the slot size of the simulations was dictated by the frame size and the number of slots per frame, the hardware specifications pose constraints on the maximum slot size and the extra control slots and switching overhead require a frame size which is larger than the sum of the data slot sizes. The data slot size is determined to be the maximum slot size allowed by the hardware, that is, 127 bytes. Note that this amount of bytes needs to be used to accommodate a two-byte CRC and a data header of nine bytes alongside the actual data. The header includes information required for the performance analysis such as the buffer size, the frame number and the value of the frame counter at the time of transmission. Taking into account the synchronization slots, the information exchange overhead, the transmission preamble and switching time per packet and the size of the data slots, the frame duration, and therefore also the superframe counter, is specified as 4242 timer tics, that is, 132.6 ms.

The few results that will be discussed denote the measured parameters according to only a single source in order to make the comparison with the simulation section. The first result is shown in Figure 5.34, which depicts the cumulative amount of received data at the root node (left y-axis) and the buffer size at the sensor node (right y-axis) for a requested bandwidth of 16 bytes per frame. The requested bandwidth

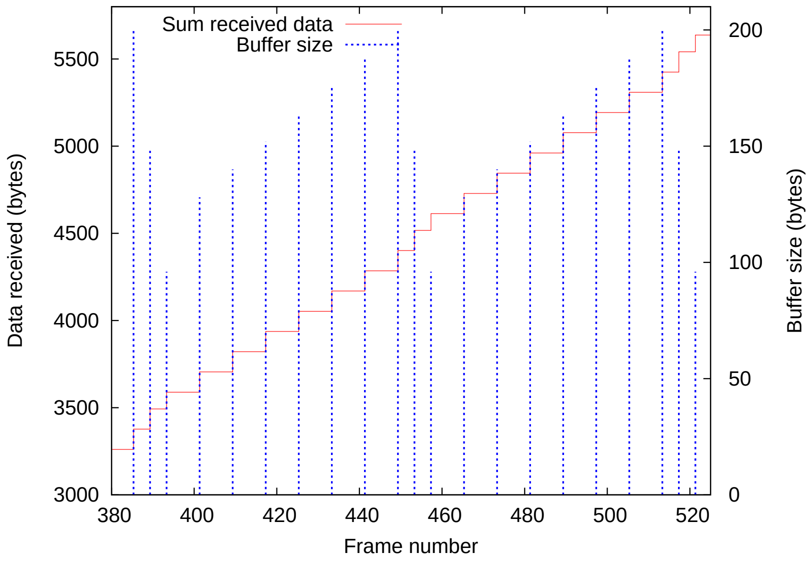


Figure 5.34: *Data arrival and buffer size at a bandwidth of 16 bytes per frame*

is approximated as a series of unit fractions $\frac{1}{8} + \frac{1}{64}$. The periodicity of the buffer size is a direct result of the scheduling protocol. The bandwidth is approximated as one fraction, which is too small, and a second fraction that compensates the difference between the requested bandwidth and the allocated bandwidth by the first fraction. This leads to a situation where the data buffer slowly accumulates until the slot, which is allocated according to the second fraction, is scheduled for transmission. The maximum buffer size that was measured is 200 bytes. When comparing the result with the equation to determine the maximum buffer size, Equation 5.28, with the given parameters, $S = 116, bw = 16, f_0 = \frac{1}{8}$ *and* $f_1 = \frac{1}{64}$, the maximum buffer size is determined to be 200 bytes. As can be noticed, even though the experimentation considers realistic switching times, overhead due to preamble, synchronization, etc. , the measured maximum buffer size can still be determined by the equation derived by means of simulations.

Since the protocol requires the aggregation of the data to use the wireless medium as efficiently as possible, the latency of the data increases. This is also shown in Figure 5.35, which depicts the latency of the data received at the sink, for a requested bandwidth of 16 bytes per frame. The resulting latency is measured by calculating the difference between the time of arrival of the data at the sink and the time the data was generated at the sensor by means of the information which was embedded in the data header. It can be noticed that the latency exhibits a similar behavior as the buffer size for the same requested bandwidth. The maximum measured latency is equal to 12.5 frames, which can also be derived from the maximum buffer size, by using the same methodology as with the simulation results. The maximum latency is deemed the time required to fill a buffer of the maximum buffer size, that is,

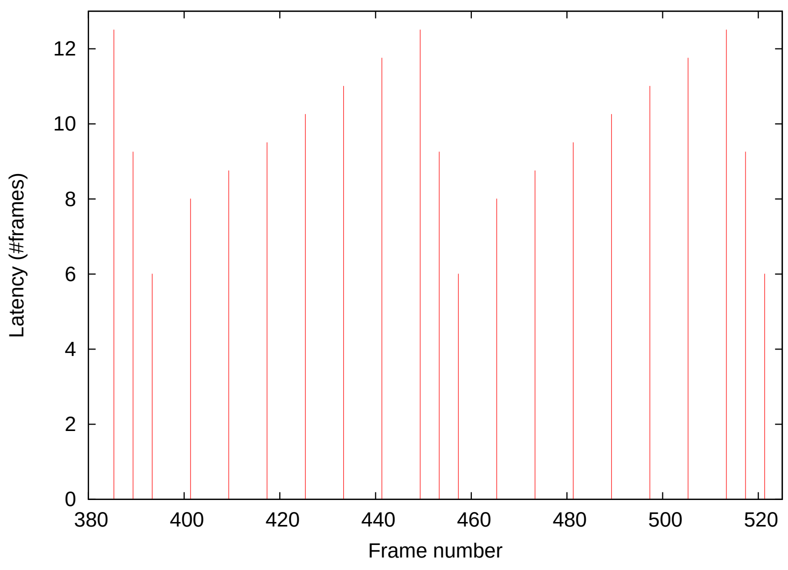


Figure 5.35: *Latency at a bandwidth of 16 bytes per frame*

200 bytes, according to the specified bandwidth, that is, 16 bytes per frame. The calculated maximum latency is 12.5 frames, which is an exact match to the maximum measured latency.

In order to be able to make the comparison with a more complex approximation, that is, an approximation consisting of more than two unit fractions, the results for a requested bandwidth of 45 bytes per frame are depicted in Figure 5.36. The results depict the cumulated amount of data that arrived at the sink (left axis) and the buffer size at the sensor node (right axis). The requested bandwidth is approximated as a series of fractions $\frac{1}{4} + \frac{1}{8} + \frac{1}{64}$. Note that a similar accumulation of data occurs as discussed in the simulation section. The bandwidth provided by the allocated slots according to the first fraction is insufficient to comply to the requested bandwidth. The remaining bandwidth, that is, the difference between the requested bandwidth and the bandwidth provided by the slot allocations according to the first fraction, in combination with the slot scheduling frequency of the second fraction results even in an excess of data, thereby requiring a third fraction to schedule slots in order to provide sufficient bandwidth. The scheduling of the last fraction is more clearly depicted in the zoomed in section of the figure at the top of the figure. The maximum bandwidth measured during the experiment is equal to 252 bytes, which also matches the maximum bandwidth calculated according to Equation 5.28.

The latency of the experiment with a bandwidth request equal to 45 bytes per frame again shows a similar behavior as the buffer size and the maximum latency measured during the experiment is equal to 5.6 frames. When deriving the maximum latency from the maximum buffer size by using the same method as in the simulation section,

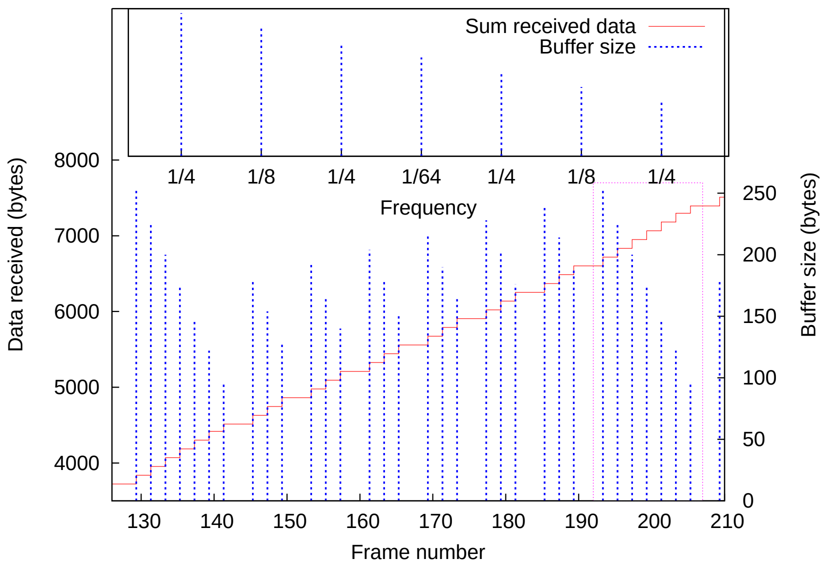


Figure 5.36: *Data arrival and buffer size at a bandwidth of 45 bytes per frame*

the maximum latency is determined to be 5.6 frames, thereby matching the obtained result.

Such measurements have been performed for every possible requested bandwidth, lower then the bandwidth provided by a single slot. Those results lead to the same conclusion, the measured maximum buffer size and maximum latency match the respectively calculated values, provided that the requested bandwidth requires at most a single slot per frame. When the requested bandwidth requires more than one slot per frame, even when one slot is a slot shared with other nodes, an increased maximum buffer size and maximum latency can be observed. Analysis of the results showed that due to the synchronization and identification slots at the start of the frame, there is an unequal distribution of the data slots. As a result, the time between the scheduled data slots might differ, since two slots scheduled in the same frame have a different interval than two slots scheduled in subsequent frames. Since the protocol maintains such a careful slot allocation, this disturbance in the balance results in an accumulation of the data during the initial slots that are dedicated to the synchronization and network joining methods. The higher the requested bandwidth, the more clearly noticeable this phenomenon is.

These results show that the protocol relies heavily on the balanced distribution of the data slots. A countermeasure that can be taken in order to ensure a correct behavior, is to place the synchronization and identification slots between the data slots, such that all data slots are equally distributed within a frame. The data transmission mechanism would consider the extra evenly distributed slots as no more than either a longer switching time or preamble tranmission time. However, it will have an impact on the maximum buffer size and maximum delay, since the bandwidth of a slot relative to the frame size decreases, when compared to the ideal case where no switching times or synchronization slots are used. But this is covered by calculating the new approximation, which will determine the new maximum buffer size and maximum delay for this fraction. In other words, the approximation will behave as predicted.

5.10 Conclusion

As could be noticed in Chapter 3, a significant number of related works, that consider a TDMA schedule, considers only a homogeneous network, that is, a network where all sensor nodes have an equal bandwidth to send their data. They allocate only a single slot per frame to any sensor node. However, in cases where both high throughput sensor nodes, such as an ECG sensor, and low throughput sensors, such as a temperature or accelerometer sensor, are present in the network, either the slot sizes become too large for the low throughput sensors or the data of the high throughput sensor becomes too fragmented to be transmitted in an efficient manner. Both methods result in an inefficient usage of the wireless medium, the former obstructing access to the medium while no data is being transmitted, the latter creating a relatively large overhead per data unit and as such reducing the effective throughput due to the extra overhead. Besides the heterogeneous character of the network, a slot allocation protocol should also attempt to keep the changes to the schedule to a minimum. Any change that needs to be propagated to the entire network results in a considerable overhead and could even result in medium access conflicts when not all nodes have noticed the updated schedule.

In the 1990's a lot of interesting network scheduling protocols were conceived, such as FQ, PGPS, Stop-and-Go queuing, etc. However, the focus of those protocols was on an entirely different problem statement, that is, a contention resolution that existed in a router or switch of a wired network. Those protocols could not dictate the behavior of their sources, they were often forced to deal with the received input and needed to ensure that all input flow received a fair portion of the bandwidth. Some of them designed the fairness in such manner, that real-time or high priority flows were assigned a certain guaranteed bandwidth, whereas best effort traffic was scheduled inbetween the real-time traffic. None of those protocols were concerned with, or could even influence, the slot allocation of the source, in order to prevent collisions. That being said, certain concepts are certainly worth considering, as well as certain slot allocation methods in other related work, provided that the information is modified for a suitable use in Wireless Sensor Networks.

This chapter discussed briefly a slot allocation method based on the greatest common divisor (gcd). The nodes are assigned a fraction of the available bandwidth, respective to the proportion of requested bandwidth to the total available bandwidth. By determining the lowest common multiple (lcm) between the denominators of all fractions, all nodes are ensured a sufficient bandwidth, independent of the other requests. The algorithm ensures that any update of the schedule does not interfere with the already existing schedule. While the protocol shows some promising results when the lcm of the requested bandwidths is near the requested bandwidths, it has the potential hazard of creating very large cycles when the gcd is low.

Based on the experience and results of the previously discussed protocol, an improved algorithm is derived which makes use of Egyptian Fractions. The protocol is designed for fairness, even when the network is crowded with high bandwidth sensors, the low bandwidth sensor should still be able to send their data. The basic idea is to allow all nodes to use the whole bandwidth provided by a single slot, but then spread out in time, such that the requested bandwidth is approximated. It is better to use the wireless medium for a small unit of time at the full bandwidth, than sending each time just a bit of data. This is also more efficient concerning overhead. The more data that is being sent at a time, the smaller the header is relative to the amount of data. In order to achieve such efficient allocation of the slots, each sensor node is expected to notify its parent of its requested bandwidth. The bandwidth request is considered as a fraction of the total available bandwidth and can be approximated by means of a series of unit fractions, consisting of only unit fractions with a denominator equal to a power of two. Such series of unit fractions is called an Egyptian Fraction. Since each unit fraction represents the frequency with which a slot is scheduled, it is sufficient to determine an initial slot and frame index to define a complete slot schedule. Note that the allocation of the slots is done for each unit fraction, starting from the unit fraction with the highest frequency. The initial slot is determined by means of the traversal of a binary tree, which can also expressed as an equation denoting that each initial slot position is situated in the middle of the periodic interval of the previous allocated fraction. Thanks to the binary tree traversal and the unit fractions consisting of denominators of a power of two, the method is straightforward, while at the same time conflicting slot allocations are avoided. Note that the heterogeneous character of the network is supported by allowing the sharing of a single slot over multiple frames. A sensor node which requires only a fraction of a slot per frame, is allowed to use the slot at its full capacity once every x frames, determined according to the unit fractions contained within the approximation, such that its requested bandwidth is supplicated. The resulting allocation indicates in which frame the slot may be used. This scheme leads to a schedule that is cyclic, determined by the lowest fraction in the approximations. As can be noticed, this protocol is not work conserving, it is also not supposed to do so. The time that no slots are scheduled, can be used to put the sensors to sleep.

On top of the already interesting properties, the protocol is also designed keeping in mind possible hardware and wireless medium constraints, thereby reducing the amount of overhead, taking into account the lossy character of the wireless medium, etc. The merging of slot allocations of different nodes occurs automatically by the slot allocation method, which prevents any allocation conflicts. Therefore, the existing slot allocation schedule is not disturbed by any node joining the network or nodes requesting extra bandwidth. Moreover, thanks to representation of the slot allocation in terms of fractions and their initial slot and frame index, the design results in a periodic schedule, allowing to sent the slot allocation scheme only once for the duration of the network. This not only reduces the overhead significantly, but also alleviates any possible scheduling conflicts due to missed schedule updates. Moreover, the schedule allows both high throughput sensor nodes and low throughput sensor nodes to coexist in a fair manner in the same network, without affecting the efficiency or network performance. An extra benefit of the fractional and binary representation is the minimal required size to represent such slot allocation, thereby reducing the overhead to send the schedule, even though it is only a one-time overhead.

The performance of the protocol is analyzed in detail by means of simulations and an implementation. First, the protocol is analyzed while considering a theoretical linear arrival rate of the data for both trivial cases and more complex approximation compositions. One of the findings that resulted from these simulations is the knowledge that the maximum buffer size is deterministic and can be calculated by means of the fractions contained within the approximation, the requested bandwidth and the slot size. Since the maximum latency is directly related to the maximum buffer size, it can also be considered as deterministic. However, note that while the maximum buffer size depends mostly on the number of fractions within the approximation, the maximum latency experiences a large impact from the requested bandwidth. As such, the maximum latency of low bandwidth requests is significantly higher than the latency of high bandwidth requests. The obtained knowledge about the maximum latency of the protocol allows to control the specific parameters, such as the approximation accuracy by bounding the maximum possible denominator within the approximation, in order to provide either a more accurate bandwidth usage, or an improved latency. Note that this control can be enforced on a per node level, allowing a single node to have a high precision in the approximation, while allowing a different node to have an excellent latency.

Since the simulation was based on a linear arrival pattern of the data, the following analysis considered the data arrival in bursts. The results obtained from the simulations indicate that a higher packet size leads to a higher maximum buffer size. As can be expected, a packet size for which the slot size is a multiple provides a lower maximum buffer size. The lower maximum buffer size experienced because of an approximation with a limited number of fractions can be found for all possible packet sizes. Interesting to note is that the arrival of data in bursts results in a periodic super-cycle in which at least one regular cycle according to the protocol is contained. The number of cycles is determined by the packet size, the duration of the cycle and the requested bandwidth. When the super-cycle consists of only a single cycle, a lower maximum buffer size is achieved. As such, certain combination of the packet size and the requested bandwidth also ensure a lower maximum buffer size.

A last simulation analysis determined the performance of the protocol within a networked environment. The fairness of the data arrival of different nodes is studied in terms of bandwidth and latency. The results showed that the protocol results in a fair bandwidth allocation, ensuring an equal distribution of the buffer size. The fairness is also reflected in the experienced latency, where it is clear that the data from all nodes experience a similar latency for a specific requested bandwidth.

While the simulations provide essential information which would otherwise be hard to obtain or take a long time to obtain, they are still only based on a model which only attempts to approach the reality as close as possible. None of the radio switching times, preamble overhead, etc. is considered during the simulations. Moreover, the network synchronization and association overhead was not taken into account. Therefore, the discussed protocol was implemented on a sensor platform in order to verify its operation in reality. Although the simulation network assumptions ignored the synchronization overhead, time required to transmit the preamble, switching times, etc., the measured latency and buffer size show a perfect match to the maximum buffer size calculations based on the equation obtained from the simulations, provided that only a single slot is scheduled per frame. Due to the unequal distribution of the data slots as a result of the synchronization and identification slots at the start of each frame, the measured values for the maximum buffer size and the maximum latency are larger that the calculated values. The flaw can be easily resolved by evenly distributing the data slots by distributing the synchronization slots and identification slots between the data slots, such that all data slots are evenly spaced, provided that the diminished slot capacity is taken into account when calculating the approximation.

Part II

Wireless Local Area Networks

CHAPTER 6

Non-standardized operation on IEEE Std 802.11 hardware

6.1 Introduction

Since the conception of IEEE Std 802.11, research has been dedicated to use Commercial Off-The-Shelf (COTS) IEEE 802.11 hardware in a non-standardized manner. The main drive behind this research has been the lack of certain features in the standard, due to which its default operation resulted in suboptimal performance in certain application domains. Some of the encountered issues, discussed in the following paragraphs, include idle listening, no QoS guarantee, that is, no real-time capabilities support, and suboptimal multicast transmissions.

The standard does provide a Power-Save Mode (PSM) to allow idle clients enter a low power mode, only waking up at certain intervals to receive the beacons of the Access Point (AP). Should the Traffic Indication Map (TIM), which is embedded in the beacon packet, announce the availability of buffered data for the client at the AP, the client needs to wake from its low power state to receive the data, addressed to the client, from the Access Point (AP). While such method prevents idle listening for idle clients, the busy clients are required to keep their radio active in order to contend for the channel, which could prove to be disastrous in power critical situations [162]. A TDMA type of operational mode would prevent idle listening and requires stations to be active solely when actually transmitting or receiving data.

WiFi is, besides its common employment in Wireless Local Area Networks (WLANs), also exploited in long-distance networks thanks to its cost-effectiveness [163]. An imperative capability of such network should be the support of a guaranteed QoS, such that real-time video conferencing would become feasible. To support such feat, a delay-bounded scheduling could be employed, that is, a TDMA type of modus operandi. Note that for long-distance networks TDMA is also required to alleviate issues during normal operation, such as the round trip time of the acknowledgement, the carrier sensing which might not detect the transmission of farther located nodes, etc. A second application environment in which real-time capabilities are required, is where industrial communication environments make the effort of migrating from wired to wireless applications. Whereas the coexistence of real-time controlled stations with regular stations was made possible by imposing traffic smoothing on the regular stations, such strategy is not possible with a wireless network, since the wireless medium is an open medium. A TDMA based approach, on the other hand, would support real-time communication services with CSMA-based networks [164].

While the IEEE Std 802.11 provides plenty of support and optimizations for unicast messaging, applications that require multicast messaging need to be satisfied with the bare minimum. The standard does not define an ACK mechanism for multicast transmissions, which reduces the possible support for multicast packets drastically. Lost multicast packets are not retransmitted due to the lack of information whether or not the packet was received successfully. Moreover, due to the lack of information whether or not a packet may have collided with another frame, the contention window (CW) is kept fixed, which is the minimum CW. Such behavior can be considered to be unfair towards other transmissions, where the CW is adjusted according to Binary Exponential Backoff algorithm. The transmission rate of multicast packets is typically fixed to a low basic rate, which not only limits the achievable throughput for multicast packets, but also results in a relatively long occupation of the wireless medium. In order to employ multicast packets in a multimedia streaming use case, either a workaround solution needs to be employed, such as the 'leader-based' multicast [165], or the operation of the MAC protocol needs to be adjusted, such that it works in a non-standardized manner.

Note that most related work focuses on either the station and/or access point functionality, whereas the ad hoc and Mesh mode operation are not considered in these types of work. For this reason, is the discussion in this chapter and the following chapter also limited to the station and access point functionality.

Thanks to the open source character of the Linux kernel and the SoftMAC approach of most drivers (see Section 6.4.4 for a detailed description), the Linux kernel is a quite natural choice to proceed with. A significant amount of the MAC functionality can be adjusted, thereby opening the possibility to use COTS hardware in a non-standardized manner. The towards this concept most relevant sections of the Linux kernel are discussed in Section 6.4. Before going into detail on the Linux kernel internals, the following section depicts some basic facts and procedures of the IEEE Std 802.11, specifically for a Basic Service Set (BSS) type of operation. Section 6.3

enumerates research where a non-standardized operation of the IEEE Std 802.11 standard has been investigated. Since certain operations require a strict timing, a description of the operation of some real-time operating systems is also provided in the last section. Note that this chapter does not contain new results. This chapter ensures that sufficient information is given such that the issues and the potential within this research can be understood. This chapter also shows some related work, indicating certain application domains of the research.

6.2 IEEE Std 802.11

The IEEE Std 802.11 defines a single Medium Access Control (MAC) and several physical layer (PHY) specifications for wireless connectivity within a local area [12]. A node comprising an IEEE Std 802.11 compatible radio can operate in a multitude of modes, often dependent on the type of network architecture the nodes are organized in. A Wireless Local Area Network (WLAN) is usually organized in a Basic Service Set (BSS). The standard defines several types of BSS, such as the infrastructure BSS, the Mesh BSS and the Independent BSS (IBSS), sometimes also referred to as an ad hoc network. A set of possible operational modes are defined to be used within each BSS type, such as Access Point (AP) or station mode in an infrastructure BSS. Note that the standard also defines operational modes that are to be used outside the context of a BSS.

The success of the IEEE Std 802.11 lies partially in its compatibility with the wired IEEE 802 LAN. Therefore, it is paramount to maintain this compatibility, the wireless device should appear to the Logical Link Control (LLC) as a wired IEEE 802 LAN. In order to achieve such feat, the MAC layer of an IEEE Std 802.11 compatible device needs to incorporate a significant number of functions. As a result of this vast functionality, augmented by the different supported modes, technologies and high throughput mechanism, the standard covers a substantial number of topics. Therefore, this section will not cover all specified functionality, the discussion will be limited to the infrastructure BSS context and will consider solely functional descriptions relevant to modifying the transmission behavior of the MAC into a more deterministic transmission pattern. The content of this section is derived from IEEE Std 802.11$^{\text{TM}}$-2012 [12].

Besides the DCF, two other coordination functions are defined: Point Coordination Function (PCF) and Hybrid Coordination Function (HCF). The PCF will not be discussed since, although it is specified in the IEEE Std 802.11, almost none of the major hardware manufacturers have provided support for this optional functionality.

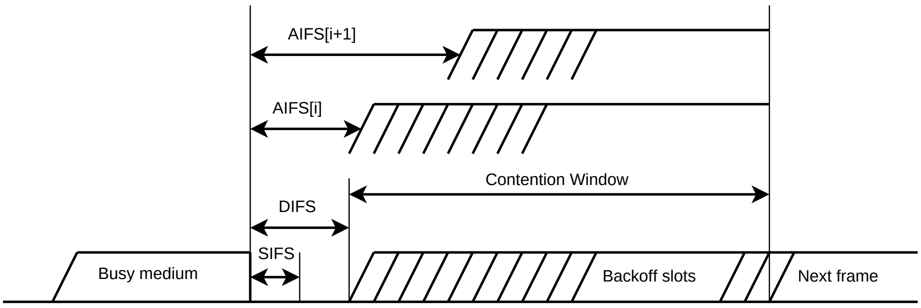


Figure 6.1: *IEEE Std 802.11-2012 Interframe space.* *(Ref: Adapted from IEEE Std 802.11^{TM}-2012, Figure 9-3, page 826.)*

6.2.1 Distributed Coordination Function (DCF)

The sharing of the wireless medium is made possible by the Distributed Coordination Function (DCF) through the use of CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance). In order to assert whether the medium is idle, the CSMA/CA protocol makes use of Clear Channel Assessment (CCA), which includes Carrier Sensing (CS) and energy detection (ED). Besides the collision avoidance, DCF, amongst others, also ensures that individually addressed traffic is immediately acknowledged by means of an ACK frame. Failure to receive this ACK frame leads to the retransmission of the data.

The standard defines a number of interframe space periods, which define the required time that needs to be considered between subsequent actions in order to be compliant with the standard. One of such interframe spaces is the DCF interframe space (DIFS), which is commonly used when STAs are attempting to access the wireless medium. A Short interframe space (SIFS) is an interframe space smaller than DIFS and is usually used in cases where the sequence of transmissions should not be interrupted by possible colliding transmissions of other stations, such as in the case of a second or subsequent MAC Protocol Data Unit (MPDU) in a fragment burst. Note that not all interframe space intervals that are specified in the standard are discussed here. The discussion is constrained to the most relevant ones.

The contention avoidance mechanism is focused around the most critical point in time, that is, after a busy period where stations attempt to resume their transmissions. In general are stations required to wait for at least a DCF interframe space (DIFS) period during which the medium is considered idle, as depicted in Figure 6.1. Note that after the reception of a frame, not destined to this STA, for which a reception error was detected or which contained a false MAC Frame Check Sequence (FCS) value, an Extended Interframe Space (EIFS) period is used instead of a DIFS period, in order to provide sufficient time to the destination to send an acknowledgement in case it received the frame correctly. After the DIFS period, or EIFS period, stations enter the backoff stage, select a Contention Window (CW) value at random between the minimum CW and maximum CW values and initialize their backoff counter with the CW value. At each backoff slot boundary, this counter is decreased if the medium is considered to be idle, determined by means of the CS, and a station is allowed to start transmission if the counter reaches zero. Should the medium be considered busy during a backoff slot, then the backoff procedure is suspended. Only after an idle medium for the duration of a DIFS period, or an EIFS period in the appropriate case, was detected, is the backoff procedure allowed to resume its operation.

In case a data frame is not received correctly, or the acknowledgement was not received correctly or timely, the sending station will increase its maximum CW value by means of the Binary Exponential Backoff (BEB) algorithm. When the maximum CW value has been reached, the CW value shall remain at this maximum value, until the CW shall be reset to the minimum CW value. The CW value is reset to the minimum CW value upon the sucessfull transmission of a frame. Besides increasing the CW upon a failed transmission, is either the Short Retry Counter (SRC) or the Long Retry Counter (LRC) incremented, in case the transmitted frame contained an MAC Protocol Data Unit (MPDU) of type Data or MAC Management Protocol Data Unit (MMPDU). The selection of which counter is incremented is determined by verifying whether the length of the data frame is above a certain predefined threshhold, which, as will be discussed later in this section, will also be used to determine whether or not a Request-to-Send/Clear-to-Send (RTS/CTS) message exchange is to be used for the frame. If the SRC becomes equal to the short retry limit or the LRC becomes equal to the long retry limit, the retransmissions stop and the packet is dropped, unless multiple rates are possible for transmitting the packet. In the latter case, the packet is retransmitted at a lower rate, until the retry threshold is reached, after which the next rate may be tried, until no more rates are specified.

Every STA shall also maintain an STA Short Retry Count (SSRC) and an STA Long Retry Count (SLRC). The SSRC and SLRC counters are incremented if an MPDU with type Data results in the incrementation of the SRC and LRC respectively. When either of these counters reaches their respective retry limit, the Contention Window is also reset to the minimum CW. The SSRC and SLRC are reset upon the succesful reception of specific frames, such as amongst others, a CTS frame in response of an RTS frame in the case of the SSRC, or the reception of an ACK frame in response to the transmission of a frame consisting of at least part of a MAC Service Data Unit (MSDU) or MMPDU for SLRC.

In case the medium was already idle for a duration longer than DIFS or $\mathrm{AIFS_n}$ in the case of a QoS station, the station may attempt to transmit immediately. After the end of each transmission, where no more frames of the same MSDU or MMPDU are expected to follow, for either Data, Management or PS_POLL Control frames, an additional backoff procedure is initiated, even when no more frames are scheduled for transmission.

It is clear from above description of the backoff algorithm that the CCA mechanism, which constitutes of CS and ED, plays a vital role in the DCF. Both physical as

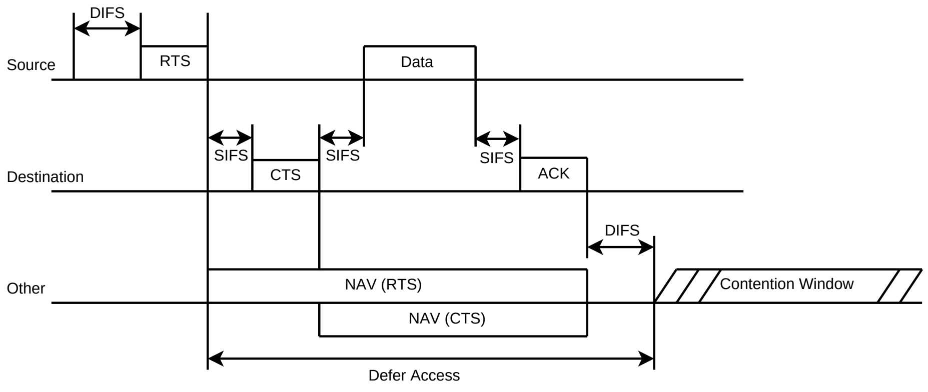


Figure 6.2: *IEEE Std 802.11 RTS/CTS/data procedure. (Ref: Adapted from IEEE Std 802.11^{TM}-2012, Figure 9-4, page 829.)*

virtual CS functions are used to determine the state of the medium. The physical CS is implemented on the NIC, while the virtual CS is provided by the MAC. Basically, virtual CS, also referred to as the Network Allocation Vector (NAV), makes use of the announcements regarding the impending use of the medium. The NAV maintains an estimation of the duration the medium is busy, based on received messages that are destined for other stations. Even when the physical CS asserts the medium is idle, no packet is transmitted when the NAV indicates a transmission is ongoing. A first method to inform stations about the duration of the current transmission is the Duration/ID field in unicast packets, where the data can be immediately transmitted after having followed the DCF procedure. Another method that can be used to perform virtual carrier sensing is the exchange of RTS and CTS frames prior to the actual data transmission, as depicted in Figure 6.2. Note that the interframe space between RTS and CTS, CTS and data, and between data and the immediate ACK is a SIFS, which is lower than the DIFS period, thereby preventing other stations to be able to interfere with the transmission of the next frame. Both the RTS and CTS frames include a duration field, which indicates for how long the medium will be busy. While the RTS/CTS exchange can relatively fast detect whether no other stations attempt a transmission at the exact same time, it also involves a certain overhead to the data transmission, as can be noticed from the figure. Moreover, the RTS/CTS mechanism can not be used in a multicast packet transmission, since there are multiple recipients for the RTS and therefore potentially concurrent senders of the CTS in response. Since the considerable overhead the RTS/CTS messages exchange introduces, not all data packets are preceded by an RTS/CTS message exchange. Packets whose length is lower than a certain defined threshold are transmitted directly, while larger packets are accompanied by the RTS and CTS frames.

6.2.2 Quality of Service

The IEEE Std 802.11e amendment introduces methods to support applications that require QoS. Within the amendment, the Hybrid Coordination Function (HCF) is specified, which can operate both in a contention-based channel access (EDCA) and by means of a centralized coordinator (HCCA). The latter, HCF Controlled Channel Access (HCCA), will not be discussed in this section, since no major hardware supplier provides support for the method. Note that this section covers the QoS provisioning in the IEEE Std 802.11 only partially, sketching only the '*Big Picture*', since only certain parts are relevant to the remainder of this work.

The channel access procedure of Enhanced Distributed Channel Access (EDCA) is derived from DCF. Instead of waiting DIFS time for the medium to be idle before starting the backoff counter, the nodes wait an Arbitration Interframe Space (AIFS) time of the specific packet Access Category (AC), depicted as $AIFS[AC]$ time. EDCA defines four categories which are enumerated here from low to high priority: Background traffic (BK), Best Effort traffic (BE), Video traffic (VI) and Voice traffic (VO). Each of those categories has its own specific AIFS, minimum CW, maximum CW and Transmission Opportunity (TXOP) value. The higher the priority of the class, the shorter the time to wait, with a minimum wait period equal to DIFS. When a node has won access to the channel after contending for it, it obtains a TXOP, that is, the TXOP specifies a maximum transmission duration. During such Transmission Opportunity, the node is allowed to send multiple packets within the assigned time period. If the TXOP value is equal to zero, the node is allowed to just send a single packet.

The different Access Categories are each controlled by their own Enhanced DCF, EDCAF, where an internal collision between the categories might occur within a node if multiple categories are queuing data at the same time. Such event is resolved by assigning the TXOP to data frames of a higher priority, while data frames of a lower priority behave as if there were an external collision on the wireless medium.

Since multiple packets are allowed to be transmitted during a single TXOP, the amendment introduces the Block Ack mechanism to optimize the acknowledgement mechanism by aggregating multiple acknowledgements into a single frame. The BA mechanism is initiated by exchanging ADDBA Request and Response frames, as depicted in Figure 6.3. During the message exchange, if the BA is accepted, the type of Block Ack, Immediate or Delayed, is determined alongside the maximum buffer size, which limits the number of frames that can be sent in a single block. If the BA setup was successful, the originator can send a series of frames, separated by a SIFS, after having obtained access to the medium, provided that the number of frames do not exceed the maximum buffer size. The recipient maintains a record of the received frames and informs the originator after it has sent the BlockAckReq frame. The originator verifies whether all frames have been received and initiates a retransmission for any frames that were not received correctly, either in another block, or individually.

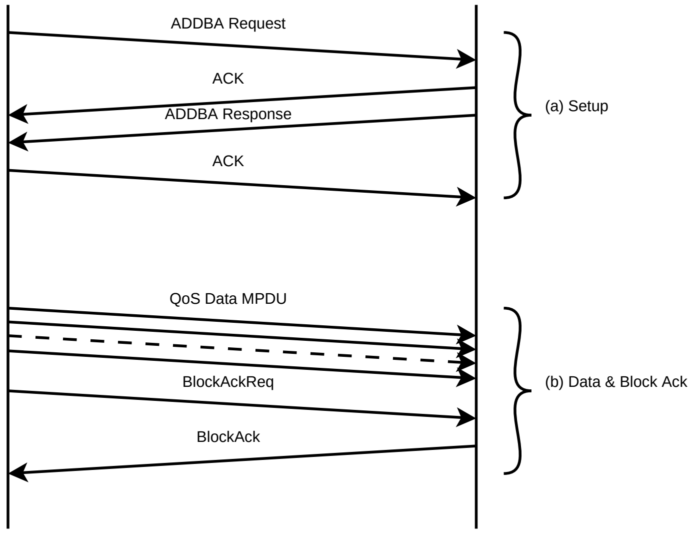


Figure 6.3: *IEEE Std 802.11 Block Ack mechanism. (Ref: Adapted from IEEE Std 802.11^{TM}-2012, Figure 9-25, page 904.)*

For QoS stations, the procedure is similar, but instead of waiting for DIFS time before entering the backoff stage, they wait for an AIFS$_n$ time, that is, Arbitration Interframe Space, which is dependent on the QoS class of the transmitted packet. The higher the priority of the class, the shorter the time to wait, with a minimum wait period equal to DIFS. A Short interframe space (SIFS) is an interframe space smaller than DIFS and is usually used in cases where the sequence of transmissions should not be interrupted by possible colliding transmissions of other stations, such as in the case of a second or subsequent MAC Protocol Data Unit (MPDU) in a fragment burst. Note that not all interframe space intervals that are specified in the standard are discussed here. The discussion is constrained to the most relevant ones.

6.2.3 Aggregation

Two types of aggregation are supported in the IEEE Std 802.11n amendment, which is part of the IEEE Std 802.11^{TM}-2012, that is, Aggregate MAC Service Data Unit (A-MSDU) and Aggregate MAC Protocol Data Unit (A-MPDU). An A-MSDU is an aggregation of frames at the MAC layer, such that the aggregation forms a single MPDU and therefore has also a single MAC header. The mechanism makes use of the property that the upper network stacks consider IEEE Std 802.11 frames as Ethernet frames, thanks to the compatibility with wired IEEE Std 802.3, which is accomplished by the IEEE Std 802.11 MAC as already mentioned in the introduction of the current section (Section 6.2). Since Ethernet headers are smaller than IEEE Std 802.11 MAC headers, the resulting MPDU will be smaller. Since the frames are aggregated within a single MAC frame, a number of conditions are placed on the frames that can be embedded within the aggregation. Such constraints include for example the requirement that DA and SA, Destination and Source address respectively, need to map to the same RA and TA, Receive and Transmit Address respectively. Moreover, all frames should have the same priority, that is, they should all map to the same QoS Access Category. Whether or not the A-MSDU can be carried in a QoS data MPDU under a Block Ack agreement is determined per Block Ack agreement. The agreement needs to indicate support for A-MSDU, otherwise no Block Ack is possible.

A-MPDU aggregation on the other hand is performed at a lower level in the networking stack, where MAC frames, that is, MPDUs, are aggregated within a single PHY frame. Each of the MPDUs is preceded by a descriptor and followed by padding bits. All MPDUs need to addressed to the same RA and there are also certain requirements involving QoS data frames. An A-MPDU needs to be transmitted within a Block Ack context or a No Ack context, since each individual MPDU needs to be acknowledged. The A-MPDU is less efficient than the A-MSDU, due to the extra overhead of each individual MAC header. However, each MPDU is accompanied by its CRC, so in the event of the reception of a corrupted frame, the MPDUs that were successfully received do not need to be retransmitted. The Block Ack mechanism ensures the sender is able to identify the frames that require retransmission.

6.2.4 Power Save Mode

As already mentioned in the introduction of this chapter, IEEE Std 802.11 specifies several ways to reduce the power consumption of a node in an Infrastructure BSS context by allowing it to enter power down mode if the network interface is idle. A non-AP station can enter Power Save Mode (PSM) by transmitting a nullfunc packet with the PM bit set to notify the AP that the station enters Power Save Mode. After sending the notification to the AP, the stations enters PSM and only wakes up at regular intervals to receive beacons originating from the AP. The AP will buffer both data frames destined to this node and broadcast frames, constrained by a certain maximum buffer size. If data is available for the station in PSM, the AP indicates in the Traffic Indication Map (TIM) field of the beacon that data is available for the station. The station is then required to wakeup and wait for the buffered data sent by the AP.

Unscheduled Automatic Power Save Delivery (U-APSD) is an enhancement of the original PSM, where in case the station send a frame of a specific Access Category, a service period is triggered and the AP can send the buffered data during that period. This discussion does not go into the details of the operation, since it is not relevant to the remainder of this work. Important is to realize the impact of such sleep cycles on the determinism of the transmission cycle.

6.2.5 Association

In a WLAN network within an Infrastructure BSS context, stations need to associate to the Access Point (AP) to be able to participate in the network. The AP does not send individually addressed packets to a station that is not listed in its associated stations list. The association procedure consists of three stages, depicted in Figure 6.4. All requests are initiated from the station that wishes to become associated. The probe request is to discover possible Access Points in its proximity, to advertise the supported rates and capabilities of the stations. An AP that receives such probe request will respond with a probe response message, indicating its SSID, supported rates, encryption types and capability information of the AP. Note that the station is not required to send probe request, in some cases it is even prohibited, the station can also wait and listen for the beacons that the APs periodically broadcast.

The goal of the authentication phase is to establish the identity of the station. Two types of authentication mechanism are supported, that is, open authentication and shared key authentication. Open authentication does not involve any significant actions, the station sends an authentication request and the AP responds. The shared key authentication involves the AP sending a challenge text, which needs to be sent back encrypted to the AP by the station.

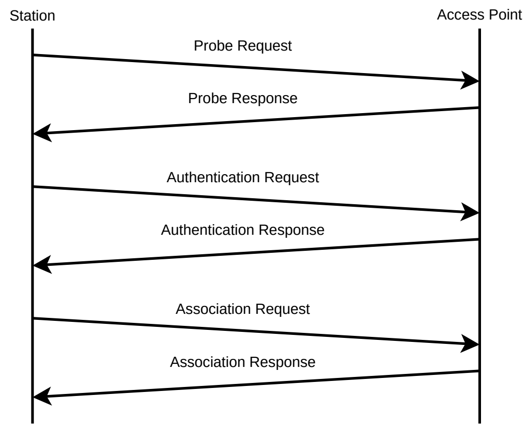


Figure 6.4: *IEEE Std 802.11 Infrastructure BSS association procedure*

Based on whether the encryption was performed with the right key, the AP grants access to the network and send an authentication response. This shared encryption key is confirmed to be vulnerable to attacks and it is therefore also dissuaded to made use of. Whereas WEP is known for its vulnerability, WPA2 is known for its secure operation and is made use of in most WLAN deployments. The authentication phase is performed by means of the open authentication method and only after the association is complete, the security authentication is triggered and a 802.1x authentication takes place, where the WPA2 authentication takes place. A more detailed description regarding authentication mechanisms is not considered, since they are beyond the scope of this work.

The last phase in the association establishment is the association phase, where the station states its requested capabilities, encryption method, etc. If those capabilities match with the AP, the AP responds with a positive acknowledgement and the station becomes associated to the AP.

6.3 Related work

As mentioned in the introduction, non-standardized operation of IEEE Std 802.11 compliant COTS hardware has become a popular research area. The specifics of the targeted deployment areas require a non-standardized operation, since either the standard does not support it, or is inefficient in this specific environment. The related work can be categorized according to their application and target specification. First the deployment of COTS hardware in WiFi-based Long-Distance (WILD) networks will be discussed, where WiFi based hardware is used to enable Internet access in remote rural areas. A second area in which a modified operation of such hardware is deemed indispensable, is where a guaranteed QoS provisioning is required. Examples of such area include VoIP, but also real-time traffic. Due to energy efficiency, the adaptation of the MAC is required in resource constrained environments, which is the area of the third category. The final category targets research that is focused on the exploration of how far beyond the boundaries of the specification of the standard it is possible to construct a reliable network. This research is mostly done in an academic context for basic research.

Since the non-standardized operation often involves a precise timing, the last section within the related work considers research in which either high resolution timers or kernel performance measurements are discussed.

6.3.1 WiFi-based Long-Distance networks

Thanks to the low cost, flexibility and ease of deployment, IEEE Std 802.11 based hardware in combination with directional antennas is popular to provide long-distance connectivity in rural areas. However, due to the specific application area, the standard operation is deficient. The propagation time of ACK messages can become thus long that the ACK timeout already expires before the message has been received. In a similar manner is the DCF mechanism lacking, since the carrier sensing might not be able to notice the ongoing transmission. Since the application field provides such challenging properties, considerable interest has been generated for it. The most interesting works are described below.

In the Digital Gangetic Plains (DGP) project, the feasibility of such WiFi-based Long-Distance network is explored [166]. The research exposed, besides the physical layer issues that needed resolving, several MAC layer issues. The IEEE Std 802.11 is designed to operate in a local area, while the considered distances in this application domain are in the range of 10 km to 100 km. As a consequence, the timeouts specified within the standard are insufficient for this application domain. The propagation delay of packets is much larger than those which are considered in the standard. As such, the ACK timeout is too small and the contention window slot time needs to be reconsidered. It was found that the most appropriate method to ensure a proper operation of the long-range testbed is a TDMA based approach. Such strategy was also implemented on the testbed, by making use of COTS hardware.

A work in which the inefficiencies of the CSMA/CA MAC of IEEE Std 802.11, such as ACK timeouts on long-distance links and unnecessary contention resolution in a point-to-point link, also led to the design of a TDMA MAC on top of COTS IEEE Std 802.11 hardware is presented in [167]. By manipulating the functionality of the CSMA/CA based hardware, a more reliable and deterministic MAC protocol is achieved. The entire system is configured to use IBSS mode, with a unique SSID for each link. To eliminate the early ACK timeout, unicast packets are converted into broadcast packets, while the reverse is done at the receiver side. While this action removes the hardware acknowledgement mechanism, it does not prevent the implementation of an ACK mechanism in software, which could take into account the long-distance characteristics and therefore decide on a block acknowledgement mechanism instead of a packet per packet acknowledgement mechanism. In order to ensure a timely transmission, a specific hardware feature was employed to disable carrier sensing and therefore backoff. When scheduling a transmission, the receive chain is configured to use an unconnected antenna connector, which therefore only measures noise. While receiving, the receive chain is reconfigured to the connected antenna. The protocol is designed for the nodes in a single link to send data until a certain quota has been reached or no more data is available, after which the role of sender and receiver is reversed. The sender transmits a marker packet upon switching its role. When the marker packet was not received at the receiver side, due to interference or other environmental occurrences, the receiver switches into the transmitter role after a timeout.

A protocol, based on the previously discussed protocol, can be found in [168], where the performance of WiFi-based Long-Distance (WiLD) networks is found to be lacking. The IEEE Std 802.11 protocol was not designed to operate in such environments and therefore shows shortcomings for this application domain. The issues involve a low bandwidth utilization due to the IEEE Std 802.11 link-level recovery mechanism, inappropriateness of CSMA/CA over long distances, due to the carrier sensing, and inter-link interference. By introducing bulk packet acknowledgements, the link utilization is improved, since a sender is not required to wait for the ACK packet to arrive for each individual packet. Due to the specified Distributed Interframe Space (DIFS) in the IEEE Std 802.11, the range for which the CSMA/CA method can be used efficiently is limited. Nodes would start transmitting packets, unaware of the packets being sent from a remote node. A TDMA type of medium access is proposed to eliminate the carrier sensing issues and to synchronize the packet transmissions and as such prevent inter-link interference. A loose synchronization approach, similar to the basic linear time synchronization protocol NTP, is chosen to evade issues that can arise in lossy environments. The adaptations were implemented on top of the driver by means of the Click modular router, which unfortunately also results in a less precise time synchronization. Besides the previous modifications, the Link Layer association is disabled by using the Atheros chipset in ad hoc demo mode, the automatic ACK transmission is disabled by using QoS frames with WMM extensions and setting the no-ACK policy and the Clear Channel Assessment (CCA) is disabled in the driver. The proposed solution led to a performance improvement in TCP throughput of 2.5 when compared to a conventional MAC.

Starting from the conjecture that CSMA/CA is insufficient in WiFi based long-distance mesh networks, the construction of a TDMA protocol is described in [169]. The MadWiFi driver has been utilized in monitor mode, such that all packets can be captured, while disabling automatic acknowledgements and RTS/CTS. Instead of using the default IEEE Std 802.11 frame format, a custom MAC header has been designed, while still maintaining compatibility with the standard frame as regards to NAV. The protocol defines a modus operandi for two types of packets; data and control packets. While the data packets traverse the complete networking stack, control packets are generated locally in the MadWiFi driver. Each control packet is assigned a timestamp to enable network-wide synchronization. The timestamping in software results in inaccuracies since the time the hardware requires to send the packet is uncertain and variable. Therefore, each control packet is assigned a timestamp by the hardware by setting a certain flag associated with this packet. The protocol exerts a precise control of the transmission times of data packets by buffering all packets that need to be transmitted, which are dequeued and processed by the regular packet transmission routine as usual during the TDMA transmission opportunity. In order to control the packet transmission opportunity, a timer function is required. The employed Atheros chipset AR5212 provides two sets of timers, which operate in a very precise manner with RX interrupts disabled, however, the performance deteriorates quickly with an increasing number of RX interrupts. The Linux software timer on the other hand has a sufficient performance in both low and high load conditions.

All previous works were in some way making use of the MadWiFi driver, which is a driver for the older generation of IEEE Std 802.11 compliant cards, i.e. IEEE Std 802.11n is not supported. The implementation of JaldiMAC, described in [170][171], refers to a WiFi-based Long-Distance network, based on IEEE Std 802.11n enabled WiFi cards, thereby opening the possibilities to use high throughput hardware. The protocol is partitioned in two parts, a click module, which resides in userspace and a kernel driver for the specific hardware. The available ath9k driver is claimed to be unstable and therefore a custom driver was developed. The precise timing is done in the kernel module, while a more crude, relative timing is defined in the click module, thereby making sure that the timing is in relative values and not in absolute timestamps in order to cope with timer jitter or unexpected transmit latencies.

6.3.2 Guaranteed QoS

A contention avoidance algorithm such as CSMA/CA as defined in IEEE Std 802.11 is not capable of providing QoS guarantees for real-time voice and video conferencing [172]. Therefore, the design of LiT MAC, a TDMA MAC, is proposed, which operates on COTS hardware. The protocol is implemented by modifying the MadWiFi driver. While the beacon transmissions are disabled, the protocol makes use of the hardware timestamping and synchronization functionalities at transmitter and receiver side respectively as regards control packets. As such, pairwise synchronization between nodes is made possible. The packet transmission control is performed by using the Linux software timer, which is more precise than the hardware timers, especially under heavy load conditions. Upon the timer trigger, the packet is placed in the transmit queue and a signal is given to the hardware to send the packet immediately. In order for the hardware to send the packet without further delay, the CCA and backoff are disabled.

The work in [173] shows that Voice over IP (VoIP) introduces a considerable capacity decrease of an IEEE Std 802.11b hotspot when six to eight VoIP stations share the medium. Neither the IEEE Std 802.11a/g nor the IEEE Std 802.11e amendment seems to resolve the issues. To remediate the problems, a software extension is proposed, which allows VoIP traffic to share the channel in a TDMA manner. This extension allows VoIP traffic to be sent without channel backoff while sensing the medium for a shorter time, thereby providing a statistically higher probability for VoIP traffic to be sent over the medium. This is achieved by manipulation of the interframe space and backoff. Since VoIP traffic can still collide with other VoIP traffic, it is scheduled in a TDMA manner, where the clients are synchronized coarsely.

A similar approach as [173] has been taken in [174], where the problem definition does not consider VoIP packets specifically, but targets real-time traffic in general. In case a real-time node is contending for the medium with standard nodes, then the real-time node should obtain the medium access. By manipulating the arbitration interframe space (AIFS) and Contention Window (CW) parameters, such feat can be achieved. Real-time traffic will use the highest priority level, that is, it will use the same AIFS values as the voice category. On the other hand, backoff is disabled for real-time traffic by setting the minimum and maximum CW to zero. Since multiple real-time nodes can still contend for the medium, thereby causing contentions, an extra upper layer is added, which serializes the transmission of real-time nodes in time. Interesting to note is that the same solution was already described in [164], in which the real-time nodes are scheduled according to a TDMA schedule in order to evade contention between real-time nodes. Within the allocated time slot, the real-time nodes still need to contend for the medium with regular nodes. By not allowing the real-time nodes to backoff, they will obtain precedence in accessing the wireless medium.

6.3.3 Academic and commercial targets

This section covers research with no immediate application domain, besides the exploring of opportunities. The targeted areas include TDMA type of medium access protocol, but also cooperative networks, increasing the modularity of the MAC, providing a flexible test platform for CSMA based MACs, and increasing the determinism of packet transmissions.

One of the works that formed an inspiration to many others, is SoftMAC [175], where the properties of IEEE Std 802.11 that should be disabled were clearly identified when working with commodity hardware in order to create a precise control over the timing of the wireless transmissions and receptions.

MadMAC [176] is a protocol that is entirely implemented in the kernel and operates on top of the MadWiFi driver. The goal is to obtain a configurable MAC which transmits packets at a configurable time without triggering the CSMA functionality of the hardware, such that the packet transmissions do not experience delays from neither contention nor backoff. In order to disable some of the CSMA functionality, the hardware is placed in monitor mode. It is not clear how the timing is realized, nor is the precision of these timings specified. FreeMAC [177] can be deemed as an extension to MadMAC, where FreeMAC focuses not only on TDMA, but also other MAC protocols that require strict control over the timing. Moreover, it extends its applicability to multichannel MAC protocols by providing the ability to switch between different frequency channels. The protocol makes use of the timer which is embedded in the WiFi card.

Instead of implementing a TDMA MAC on COTS hardware, the research in [178] was focused on defining a flexible MAC architecture where the testing of CSMA MACs would be facilitated. The implementation was performed on top of the MadWiFi driver. The transmission control allows only a single packet in the hardware queue, hence the buffering of incoming packets from the kernel in the framework. The frame scheduler is responsible for delivering the designated packets to the hardware, based on the schedule of transmissions. Instead of employing the Linux scheduler, a tasklet is recursively scheduled which reads and compares the current timestamp value of the NIC. In such manner a precise timing control should be obtained. According to the results, the performance of FlexMAC is very close to that of hardware implementations.

Soft-TDMAC [179][180] is a protocol, where all pairs of network clocks are said to be synchronized to within microseconds of each other. By considering the network as a collection of pairs, they achieve a network wide synchronization. The protocol consists of three parts, a small kernel module, a userspace library and a userspace application. The kernel module mainly forwards incoming and outgoing packets to and from the userspace library and configures the physical network card to abandon its default CSMA/CA behavior by relying in the IEEE Std 802.11 QoS features of the driver. All MAC relevant functionality is contained in the Soft-TDMAC application, which is linked to the user-space library. The library itself is responsible for the starting of timers and the triggering of Soft-TDMAC upon the elapse of such timer. One of the arguments for this split functionality is the avoidance of difficult kernel programming, since most lies in userspace. However, since most of the functionality lies in userspace, means also that it is susceptible to all kernel functionality, since it has a higher priority over all userspace functionality. Such design could deteriorate easily when the load of the system increases. To avoid such issues, the authors decided to use the Linux RT patch, which allows to tweak the real-time priorities of the running processes, including kernel processes.

As the research in [181] shows, the Linux kernel provides multiple possibilities to modify the MAC behavior of COTS hardware. Unlike previously discussed work, which adjusted the MadWiFi and other drivers, which are hardware specific drivers, this work implements cooperative retransmission functionality in the SoftMAC, which is the mac80211 kernel module (see Section 6.4.4) that implements a significant amount of IEEE Std 802.11 MAC functionality and is used by a multitude of hardware specific drivers. The design of the protocol has been implemented in the kernel module, based on the Linux kernel version 2.6.32. The added functionality places unicast data or management frames, that is not directed to the current node, into a buffer. At the time of storing the packet, a timer is scheduled for the duration of ACK timeouts. Upon overhearing an ACK message for the buffered packet, the packet is dropped and the timer is canceled. However, if no ACK has been overheard for the buffered packet upon the elapse of the timer, the node sends the buffered frame on behalf of the original sender node. Broadcast and multicast frames are ignored by the added functionality and passed on to the upper layers, thereby allowing the relay node to operate the same as a regular node and become associated. The precise timing of the ACK timeout is achieved by employing the high-resolution timers, hrtimers.

Like in the previously discussed work, the work in [182] also modifies the Linux kernel mac80211 module, that is, the SoftMAC. The work claims to improve the modularity, flexibility and virtualization of the IEEE Std 802.11 MAC functionality in the Linux kernel. The mac80211 kernel module is a monolithic block, which supports a significant number of functionalities, such as high-throughput (HT) and MAC Layer Management Entity (MLME). Both these functionalities are modified in a similar manner as the rate control algorithm, which depend on the mac80211, but can be build separately as a kernel module. Two auxiliary kernel modules are added to the MAC functionality, the Service Scheduler and the Function Handler, which allow for the scheduling and registration of the new services. The execution of those new services can be triggered by certain events, such as packet reception, packet transmission, channel switching, etc.

The following invention [183] specifies a method for a communication technique which facilitates the transmission of data packets in IEEE Std 802.11 frames by means of an IEEE Std 802.11 compliant chipset using TDMA. Instead of forwarding packets directly to the Network Interface Card (NIC), packets are queued in a software queue. The data packet is forwarded to the hardware queue, where the transmission of the packet is triggered by time-slot control information. The control information can include a count value for a timer or a timing signal generated by a software timer. In order to ensure a timely transmission of the scheduled packet, contention avoidance may be disabled or the hardware may be configured to transmit only a single packet at a time.

The authors of [184] developed a protocol, RT-WiFi, where the IEEE Std 802.11 MAC and network card driver is replaced by their own implementation of a MAC. By using a timer as a reference for the link scheduler, tasks and data can be reliably scheduled. While the protocol seems to work fine in an isolated area, the authors noticed that there is an issue when operating in a non isolated area, where other WiFi devices are available, which is an issue that is not discussed in any of the other papers, whereas it jeopardizes for example the synchronization of the entire network.

The framework proposed in [185] is not limited to the modification of a Linux driver or the mac80211 module, it replaces the complete MAC mechanism. The framework allows the dynamic and flexible configuration of a MAC protocol, either being TDMA or CSMA based. The firmware of commodity hardware is rewritten by means of the open firmware source code and accepts new MAC configurations, which are then executed. Such method is only applicable to hardware where the firmware is allowed to be replaced, i.e. it depends whether the method is public knowledge. However, although such an implementation would require a significant amount of work, the results would be very accurate since there is no dependency on the timings of the host system.

6.3.4 Power Efficiency

Use cases such as compressed video transmissions on resource and power constrained devices need special attention [162]. While a significant amount of effort has been invested to reduce the power consumption of the compression algorithm of the video, merely transmitting this video stream comprised already one third of the total power consumption. Even while IEEE Std 802.11 proposes a power save mode (PSM), where the power consumption of idle nodes is greatly reduced, it proves to be insufficient for busy nodes. Therefore, the discussed paper proposes a TDMA type of medium access, where the Access Point (AP) transmits a schedule within the beacon message, indicating the slots of the clients. Clients can only send or receive packets during their allocated time slot, allowing them to power down during the other periods of time. In this work three different methods have been considered to implement such TDMA mode, a user level application, a traffic shaper and direct manipulation of the network driver. The first method consists of a client application, which verifies when the node is allowed to transmit data before writing to its socket. Due to the buffering at different layers in the network stack, this method fails to provide a precise accuracy. The second approach makes use of a traffic shaper, switching the allowed throughput back and forth between zero and infinity. Due to the slow responsiveness of the traffic shaper, this solution was not workable. The third method consisted of modifying the network driver directly and implementing a flag which could be set and reset by means of a system call, indicating whether it is allowed to send or not. The performance was esteemed to be similar to the application layer performance. It is believed that the buffering of packets is the root cause of this issue. In all three cases, the accuracy of the system timer was an issue. The Linux kernel version 2.4 supported only a tick timer of 10 ms, thereby providing a very coarse timer resolution.

6.3.5 Kernel performance analysis

The work that forms the base of the present timer structure, that is, the hrtimer, in the Linux kernel is [186]. The work discusses besides the structure of the hrtimer, also the structure of the clockevents and clocksources. Clockevents and clocksources provide a general abstraction of the timer hardware, enabling system timers, such as the hrtimer, to make use of the hardware timers. A performance section indicates what latencies can be expected.

In order to discover problems regarding the latency of the high resolution timer of Linux, the KTAS methodology is proposed in [187], which allows developers to determine the cause of timer latencies. An example analysis is provided throughout the paper, analyzing the performance of the hrtimer in terms of latency. It is claimed that the most prominent cause for timer latencies is the softirqs caused by the processing of network packets. By reversing the priorities of those softirqs with the priority of the timer softirq, a considerate amount of problems should be solved. Tools such as KTAS can be helpful in determining issues and improving the performance of the high resolution timers when required.

The work in [188] considers analyzing the interrupt latencies, specific to a Digital Signal Processing application. The interrupt handling latency is a conglomeration of seven components, such as the interrupt checking latency, the maskable interrupt latency, the Programmable Interrupt Controller (PIC) queue latency, etc. By placing time markers in specific locations in the kernel code, the latency of the specific routines can be deduced. A data structure is created to store the time markers and two system calls are defined which start and end a series of measurements respectively. At the end of a measurement, the kernel data structure is transfered from kernel-space to user-space, where the management process can process the information.

6.4 Linux kernel

The Linux kernel forms in combination with a library, usually the GNU C Library, and a set of user programs, in most cases GNU programs, the GNU/Linux Operating System, later referred to as the Linux system. What makes the Linux system interesting is in the first place the open source policy. It enables research where other OSs block out all access to the source code. Thanks to the vast Linux community, the code has reached a considerable level of quality and reliability. Moreover, significant effort has been invested and is still being performed to include support for a wide range of hardware devices. All those characteristics make the Linux system an ideal system for a performing research towards a non-standardized operation on IEEE Std 802.11 compatible COTS hardware.

The Linux kernel is known to be monolithic, although it supports loadable kernel modules, that is, sections of kernel code that can be included and removed while the kernel is running. Such loadable module support makes debugging an easier task, since the faulty module can be unloaded and replaced with a newer version. On the other hand, it ensures support for a huge number of hardware devices, in the form of drivers, while still maintaining a rather small kernel core. The specific driver modules only need to be loaded when the specific hardware is available. Besides the hardware support, the kernel also incorporates a rather extensive networking stack with support for a multitude of network protocols, ranging from well known network protocols such as TCP/IP and Ethernet to amateur radio networks. Some basic functionalities such as a virtual filesystem, timer functionality, memory management, processes and process scheduling, and interrupt handling are also provisioned in the kernel.

This section discusses in more detail the process management, more specifically in the context of the interaction with interrupts and tasklets, the kernel timer functionality and the structure of the WiFi networking stack, specifically oriented towards the cooperation with the ath9k driver. This section only provides the basic required knowledge to understand the subtle differences in performance of the different cases. A more thorough discussion can be found in [189],[190] and [191], which were used as a reference to compose this section. However, the most accurate and up-to-date information can be found in [192].

The Linux kernel discussion is based on the latest kernel version used in this work, that is, 3.13.9. Where appropriate, a reference is made to the specific operating functionality of earlier kernel versions.

6.4.1 Process management

One of the key tasks of the Linux kernel is to schedule the runnable processes. A process is an instance of a program in execution but is not limited to the executing code. It consists of the set of resources that are being used, such as file descriptors, an address space, global variables, etc. All processes have a similar view of the memory and CPU(s) thanks to the virtualization of both, which allows them to think they are the sole owners of those resources. It is possible for a process to contain more than a single thread, where each thread contains its own program counter, process stack and set of processor registers, but share the same address space. Threads are used to allow programs or sections of programs to perform work in parallel from the viewpoint of the user, while in reality the execution of the threads is intermingled with each other. The interesting approach the Linux kernel has opted for, is the lack of differentiation between a process and a thread. To the kernel, both appear as a normal process.

A process is described in the kernel by a *process descriptor*, of type struct task_struct, and is stored in a circular doubly linked list, called the *task list*. When a process executes a system call, it enters kernel space. The kernel executes the requested section of code on behalf of the user process. While executing in kernel space, the stack pointer is rearranged such that it points to the kernel stack of the process, which is a different memory region than the user space stack.

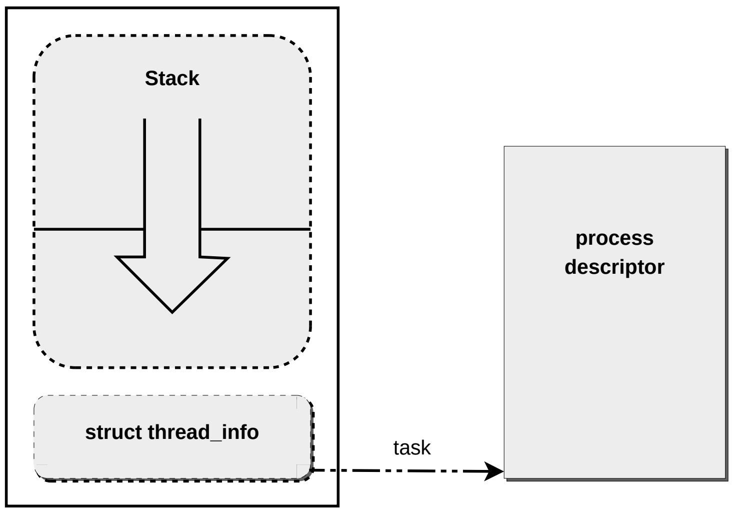


Figure 6.5: *Allocated kernel memory for a process. (Ref: Adapted from Linux Kernel Development, 3rd Edition, R. Love, Addison-Wesley, 2010 [113], Figure 3.2, page 26.)*

The execution in the kernel does not require too much stack memory, which justifies a limited memory allocation. The allocated kernel memory, depicted in Figure 6.5, serves both as stack memory and a small struct called thread_info, which is located at the beginning of the stack memory. This structure contains limited info regarding the process and points to the process descriptor. The positioning of the structure is interesting, since the kernel is able to deduce the address of thread_info struct, and therefore indirectly also of the process descriptor, by masking a number of lowest significant bits of the current stack pointer when operating in the process context. Such mechanism allows the kernel to make a reference to the *current* process, i.e, the process that is currently being executed.

One of the services of the kernel is to provide an abstraction towards the processes, such that from the view point of the process, it is the sole running process in the system. The process scheduler portrays a vital role in providing this service. It enables processes to run seemingly simultaneous, while in reality they are each scheduled sequentially during a certain time. The scheduler in Linux is preemption based, that is, when appropriate, the currently running process can be preempted in favor of another process. The algorithm to determine whether or not a process needs to be preempted takes several parameters into account. One of those parameters is the priority of the process. Each process is assigned a certain static priority, which can be either a '*nice value*' or a real-time priority. A nice value ranges from -20 to + 19, and the real-time priority ranges from 0 to 99. The nice value defines a proportional priority between normal processes, while any real-time priority exceeds the priority of normal processes.

During the 2.5 kernel development versions the O(1) scheduler was introduced, which was designed to enable the scheduling of processes in a time that is irrespective of the number of processes. The scheduler resolved an important issue of the previous scheduler, that is, it ensured that the scheduler was not required to iterate through the entire task ist in order to identify the next task that is eligible to be activated. Key to its operation is the employment of runqueues per processor. It contains two priority arrays, that each provide a single queue of runnable processes for every possible process priority. One of those arrays is designated as the active array, while the other one is the expired array. Runnable processes reside in the active priority list, unless their time slice has been exhausted. A process can be comprised in only a single runqueue, that is, the runqueue of the CPU on which the process is scheduled. The scheduling algorithm selects the highest priority process to run next until it either exhausts its time slice, the time slice counter is decreased at every clock tick, or yields the processor. If a process exhausts its allocated time slice, it is transferred to the expired priority array and is not allowed to become active again until all other processes have exhausted their time slice. Eventually all processes will become expired and the expired array can become the active array. Note that processes with a real-time priority are never placed in the expired array. The kernel provides mechanisms such that processes in the expired array do not become starved in case processes with a real-time priority are present.

The allocation of the time slices and dynamic priority ensures a careful balance between I/O bound processes that do not run a lot, but require a fast response time, and CPU bound processes that prefer to run for a long time. Processes that require a fast response time are allocated a higher dynamic priority and a larger time slice than processes that are CPU bound. The size of the time slice is determined by the static priority of a process, higher priorities are assigned a larger time slice. The dynamic priority of a process is determined by its static priority and a bonus value, ranging from -5 to $+5$, the scheduler assigns to the process based upon its previous behavior. Processes that exhibit a highly interactive behavior are 'rewarded' by increasing their dynamic priority, while CPU bound processes obtain a lower priority. The amount of interactivity of a process is determined by the amount of time it spent sleeping. This time is contained in a field of the task_struct, which is incremented by the time it slept each time the process becomes runnable again and decremented each jiffie while running. A special case is a process with a real-time priority, which can be scheduled according to two types of classes, SCHED_FIFO and SCHED_RR. The former does not keep track of its consumed time slice, it keeps running until it yields voluntarily the CPU or it is preempted by a process with a higher real-time priority. A process that is scheduled according to SCHED_RR is scheduled Round-Robin with processes of the same real-time priority, where the scheduling interval is determined by their time slices. However, it can also be preempted only by a process with a higher real-time priority.

The preemption mechanism itself is based on the dynamic behavior of the kernel and system calls. The kernel verifies at specific places in its code whether a reschedule is required, such as when the time slice of the currently running process becomes exhausted. For example, every scheduler tick, that is, every jiffie, the kernel checks whether the conditions comply to validate such action. If a reschedule is required, the flag need_resched is set. Upon returning from interrupt context or returning to user-space, this flag is checked and if needed the schedule function is called. Note that even a kernel process can be preempted, provided that it is safe to reschedule. This condition is satisfied if the kernel does not hold any locks. Therefore, each process maintains a preempt_count variable in the thread_info structure, which denotes the number of locks being held. If this counter equals zero, the kernel process can be preempted.

Although the O(1) scheduler performed very well and was used on systems with a multitude of processors, certain flaws became clear regarding the scheduling of latency-sensitive applications, also called interactive processes, where user interaction was required [191][193]. One of the issues with the O(1) is the concept of nice values and their direct relation to a certain timeslice. Low priority tasks are assigned a small timeslice, while higher priority tasks are assigned a large timeslice. As a result, if there are only low priority tasks in the system, the scheduler is forced to switch frequently from one tasks to another, whereas in the other case, the scheduler is switching very rarely to the higher priority tasks. Moreover, the difference in timeslices between two nice values is fixed, resulting in a huge relative difference between two high priority tasks and two low priority tasks, due to the fact that the difference is relatively considered a lot smaller to the timeslice of a high priority task than the one of a low priority task. Due to a suboptimal performance of the O(1) scheduler concerning latency sensitive applications, a new type of scheduler is introduced since kernel version 2.6.23. The new scheduler is called CFS (Completely Fair Scheduler) and employs principles from the fair queuing theory. The CFS scheduler is applicable solely to processes with a regular priority. The operation as previously described for processes with a real-time priority remains the same. The principle of CFS is based on the assumption that each process should obtain an equal amount of the CPU at the same time. Since in practice, such feat is not possible, CFS serializes the execution of processes. Unlike in the O(1) scheduler, which defines time slices based upon their static priority, CFS defines a targeted latency and allows the static priority, which determines its weight, to determine a proportional relation between the processes execution time. The amount of time a process can run is the targeted latency divided by the number of active processes.

This time is weighted by the relative difference between the nice values of the processes. The order in which the processes are scheduled is dependent on the amount of time they have been running. The process that has run the least, modified by its weight, is executed first. Therefore, CFS makes use of a Red-Black tree, instead of a runqueue, ordered according to the vruntime (virtual runtime) of a process. The virtual runtime of a process represents the actual runtime of a process, expressed in nanoseconds, normalized by the total weight of the number of runnable processes.

6.4.2 Interrupt handling

The peripheral hardware occasionally needs to inform the CPU of certain events. The implementation in hardware of such interrupt signals is by pulling a hardware line, which is usually connected to an interrupt controller, up or down. The CPU needs to process the interrupt by signaling the OS, which runs the respective interrupt handler. The advantage of this mechanism is the elimination of the processor needing to constantly poll the peripherals to check whether an event has occurred, thereby making more efficient usage of the CPU. Moreover, if a time critical event occurs, the processor is able to handle the event immediately.

In the Linux kernel, the device interrupt handlers reside usually in the device drivers, which are discussed in detail in [194]. The hardware interrupts are usually asynchronous with respect to the CPU clock, and can therefore interrupt the currently running process at any time. The execution of the interrupt handler takes precedence over all other kernel tasks, unless interrupts are disabled, such as in certain critical sections. Note that the interrupt handler is not associated to some process and therefore runs in interrupt context. As a consequence, a single interrupt stack is provided per CPU, they are not allowed to go to sleep, which also prevents them from using regular semaphores to identify critical sections that need to be locked. The kernel defines special types of locks that can be used in interrupt context, such as spin locks [191].

During the execution of the interrupt handler, the corresponding interrupt line is masked out on all processors. Therefore, interrupt handlers should execute as fast as possible and usually have a split functionality, a top half and a bottom half. The top half, that is, the interrupt handler, should only perform the most time critical tasks. The bottom half can run at a later time, when more appropriate, and performs the remaining tasks of the interrupt handler. The Linux kernel defines three methods to implement a bottom handler: softirq, tasklet and workqueue.

A softirq is a static method that needs to be defined at compilation time. Dynamic allocation and destruction of such object is not possible. The structure that holds the softirq handlers is a statically defined array containing function pointers. In the kernel version 3.13.9, ten different softirqs are defined, amongst others for the networking and timer subsystem. Two softirqs are reserved for tasklet activation, which are discussed in the next paragraph. When a softirq needs to be scheduled, a flag is set, indicating the availability of a scheduled softirq. Upon checking this flag, such as when returning from an interrupt, the CPU handles all scheduled softirqs in order of priority. Softirqs are not associated to some process and are therefore executed in interrupt context.

Whereas the same softirq can be run on multiple processors at once, if scheduled multiple times, only a single instance of the same tasklet is runnable at the same time. Such simplification avoids the need to provide locks to protect shared data. Tasklets can also be allocated dynamically and destroyed at run-time. Two lists of scheduled tasklets, a high priority and a normal priority tasklet list, is maintained in per-cpu data, which is traversed by the corresponding softirq handler. As a consequence, a tasklet is always executed on the CPU upon which it was scheduled.

The kernel also defines workqueues, where work can be deferred towards a bottom half running in process context, allowing it to sleep. The deferred work is executed in a kernel thread, which can be either a custom thread, or a default kernel thread.

6.4.3 Timers

Section 6.4.1 already indicated that the scheduler tick is a significant tool for most process accounting. The scheduler tick is one of the functions the Linux kernel performs every timer tick, which is used as a time reference for the whole system [194][190][191]. One tick of the system clock has a duration of one jiffy, which is $1/HZ$ seconds, with HZ being a kernel configuration parameter that can be set to 100, 250, 300 or 1000 in kernel version 3.13.9. The kernel is programmed to activate a software timer that fires every $1/HZ$ seconds to update the tick count and the wall clock time, perform process accounting, check whether there are timers that have expired or are about to expire, call the scheduler tick function and make sure the required POSIX timers are triggered. A more detailed discussion regarding those actions follows later in this section.

The static generation of ticks every $1/HZ$ seconds results in a certain overhead that needs to be handled by the CPU, especially when it is considered idle, that is, it has no process to execute. In order to prevent an idle CPU to wakeup each time a tick needs to be handled, dynamic ticks were introduced in kernel version 2.6.21 [195][196]. Such a provision allows CPUs to keep sleeping until woken up by the next timer interrupt that is not a tick. Upon leaving the idle state, the periodic tick is resumed and the number of elapsed jiffies during the idle time is updated. This ensures a better and more efficient energy usage.

Before considering the possible software timer structures in the Linux kernel, clocksources and clockevents need to be discussed, which form the interface between the hardware timers and the software timers. Depending on the type of hardware platform, a wide variety of hardware timers is available, such as acpi_pm, hpet and tsc on a standard x86 platform. Some of these hardware timers are simple counters that do not generate any interrupts, such as the tsc (Timestamp Counter). Others are more sophisticated and can be programmed to trigger an interrupt at the required time, such as the hpet (High Precision Event Timer). The Linux kernel provides an abstraction of all timers by means of clocksources and clockevents. A hardware timer that is configured to operate as a clocksource is a free running timer that does not generate any interrupts. Its current value is read out at regular times and is used to provide a reference for the system time. Clockevents on the other hand are suitable only for hardware timers that can be programmed and trigger an interrupt upon elapsing.

The selection of available clocksources and clockevents is done at boot time, where each code section, that is responsible to interface to the respective hardware timers, registers its corresponding timer as a possible clocksource or clockevent. The kernel orders the possible clocksources according to their stability and rating, and selects the most suitable timer as the clocksource for the entire system. Whereas there is only a single clocksource, each CPU is assigned a clockevent, taking into account CPU affinity, resolution and stability. A single processor system therefore only uses a single clocksource and a single clockevent. The clocksource is used as a reference and the clockevent is used to schedule a timer interrupt at specific times.

The updating of the wall time, which is done every periodic tick, is based on the operation of the clocksource. The difference between the value of the clocksource during the last update and the current value is used to determine the current value of the wall time. Likewise is the current system time calculated by taking the wall clock time, which represents the actual time of the day up until the last update, and adding the difference between the value of the clocksource during the last wall clock time update and its current value.

Part of the timer functionality, such as the timer wheel, also depends on the periodic tick count in order to detect whether timer handlers need to be run. The timer wheel, which is already available since the early versions of the Linux kernel, triggers timer handlers based on their expiry time in jiffies [197]. The timer wheel consists of five categories in which future timer expiries are placed. Each of the categories represents $256 * 64^n$ jiffies, with $n = 0 \dots 4$, where n is the category index. The lowest category represents the earliest 256 jiffies, all sorted, while each of the following categories is subdivided in 64 buckets, which are coarsely sorted. A new timer expiry is placed into the associated bucket, based upon its expiry time. Each tick, the lowest category is checked for possible timers that should be triggered. After the elapsing of the time of a lower category, the next bucket of the higher category is cascaded down into the lower category. This approach results in a high processing overhead when timers are cascaded to the lower category [198]. If the timers are removed from the timer wheel before expiry, however, the processing overhead of cascading the timer is removed. Therefore, this type of structure is useful for timers that are needed for some kind of deadline, in case an unexpected event occurred.

Note that no hardware timer source is referenced to, since the timer wheel's operation depends on the tick rate. As previously mentioned, one of the tasks that is performed every $1/HZ$ seconds, that is, every tick, is the timer wheel function. This function checks whether any deadlines of the timers in the timer wheel have expired and need their handlers to be executed. Therefore, the resolution of the timer is limited by the HZ configuration at kernel compile time.

The Linux kernel provides two types of software timers, the timer wheel and the hrtimer. The hrtimer (High Resolution Timer) [186] originates from the high resolution patch [199], which was part of the RT patch [200], and was merged into the 2.6.16 kernel [196]. Unlike the timer wheel, the resolution of the hrtimer is not limited to jiffies. It handles the timer interrupt of the selected clockevent directly, enabling a far higher resolution. The internal structure of the timer consists of a Red-Black Tree in which all scheduled timers are sorted according to their expiry time. There exists several such Red-Black trees per CPU, one for each time base. The time base determines which system time is used as a reference to determine whether the timer has already elapsed. The best known time bases are CLOCK_REALTIME and CLOCK_MONOTONIC. The former time base represents the time-of-day time, which can jump back and forth as the system time-of-day clock is changed, including by NTP. The latter represents the wall clock time, which is the elapsed time since some point in time, which is usually the system boot time unless some persistent battery backed up clock is available. The CLOCK_MONOTONIC time base is unaffected by the system clock changes and is therefore best suited to use for a performance analysis.

The high resolution of the hrtimer is assured by its methodology. The deadline of the timer that is to expire next is programmed into the clockevent that is assigned to the current CPU. Since most hardware timers generate an interrupt upon overflow, the difference between the deadline and the current time is subtracted from the maximum hardware timer value, and is programmed into the clockevent. As a result, the specific hardware timer generates a hardware interrupt at exactly the expiry time of the scheduled timer, which is directly handled by the hrtimer interrupt handler in interrupt context. The timer handler function is directly called from the interrupt handler and therefore executes also in interrupt context. In order to optimize the performance, the timer expiry can be specified as a time range during which the expiration might occur. The timer is always scheduled according to the latest expiry time, however, the hrtimer interrupt routine and the hrtimer softirq routine iterates over the list of timers and check whether their soft expiry time has already elapsed and run the timer handler when appropriate.

When the hrtimer is configured into the kernel during compilation time and is operating in high resolution mode, the tick timer is just one of the many timers the hrtimer structure needs to handle. Upon expiration of the tick timer, its periodic tasks are executed, such as process accounting and checking the timer wheel timers. If the hrtimer is not configured into the kernel or when the hrtimer is operating in a low resolution mode, such as upon boot time, the tick timer is either triggered from the periodic tick timer or from the dynamic tick functionality, depending whether the hardware timer supports one-shot functionality or can only operate in a periodic manner. The timers of the hrtimer in low resolution mode are then checked for expiration every jiffie like the timers in the timer wheel.

It is important to consider that all time related functions of the system are scheduled onto a select group of hardware timers, even just a single hardware timer in the case of a single CPU system. Note that time related functions of the system also includes every delayed task, user-space timers, wait queues, etc.

6.4.4 Networking architecture

The Linux kernel is noted for its secure and reliable networking operation. Key to this reputation is its network architecture, which will be discussed in some detail here. Since the networking subsystem contains a wide variety of protocols, its code repository has expanded to around 660000 lines of code in Linux kernel version 3.13.9, even without taking into account all the drivers for the specific devices. In order to keep the discussion within reasonable boundaries, this section is focused solely to the L2 layer of the wireless subsystem, and more specifically the AP and station mode operation. To keep the discussion tangible, the specifically considered use case makes use of a wireless card with an Atheros chipset AR9220, which is controlled by the ath9k kernel driver. Although this section is focused on the operation of WiFi network cards, the core networking operation also applies to other types of networks and some of them will be included briefly in the discussion to highlight the differences.

The Linux networking subsystem is a vast and complex matter , which is a rapidly evolving subsystem on top of that. While some references provide certain basic information regarding device drivers and partially about the networking subsystem [194][201], the provided information remains too superficial to actually matter. The available works that focus solely on the networking subsystem on the other hand[202][203], can go into more detail regarding the general architecture of the networking subsystem. However, even those specialized works only superficially consider the wireless networking subsystem; most discussions are concentrated on the L3 layer, more specifically IPv4 and IPv6. Therefore, due to the complexity and rapidly evolving nature of the wireless subsystem, the only real source of information is the kernel source [192] and the wireless subsystem Wiki [204]. The discussion in this section is based on the source of kernel version 3.13.9.

The Linux kernel networking subsystem provides support for layer L2, L3 and L4 of the network stack. An abstraction is provided for both the L2 and L3 layer; the networking core maintains a list or hash of all available L2 devices and L3 protocols.

Note that a virtual interface, such as an IPv4 tunnel interface can also be considered as an L2 device. All such L2 devices, both real and virtual, are represented by a *struct net_ device*, which represent the devices that can be detected by userspace tools such as ifconfig. The device driver allocates and registers such network device, specifying at the same time which driver functions correspond to some predefined operations, such as *ndo_ open*, that can be called when the networking core is operating on such network device. There are some exceptions, that is, devices that are not represented by a *struct net_ device* structure, such as the Bluetooth device. The lower layers of the Bluetooth stack, that is, the HCI layer, provide an abstraction by means of an '*hci device*', which is then directed towards the specific interface towards the hardware. Only when the Bluetooth BNEP protocol (Bluetooth Network Encapsulation Protocol), which allows the transportation of common networking protocols over Bluetooth [205], is used, such as for example by the Personal Area Networking Profile (PAN), a *struct net_ device* is registered to the networking core.

The differentiation between L3 and L4 is not so clear in the kernel networking subsystem, since the transport layers are affixed to only specific L3 layers. For example, the UDP and TCP transport protocols are registered only by the IPv4 protocol, while a different UDP and TCP protocol, which have been adjusted to operate specifically with IPv6, are registered to the IPv6 protocol. Thanks to the flexibility of the networking subsystem, it is possible to address any of the layers from userspace. Such interface is formed by means of sockets, which provide a certain interface to a specific protocol based upon the type of the socket. The operating parameters of the network and transport protocols can be adjusted by means of ioctl calls through the associated sockets. Obtaining and manipulating information, either device specific or general information, such as available network devices or protocols, is also done by means of ioctl calls. If the state of a device changes by executing the command associated with the ioctl call, then the networking core notifies all interested parties by means of notification chains, specific to the netdevice. In this manner, the routing layer is for example informed of a device that is disabled, such that its routing table can be adjusted.

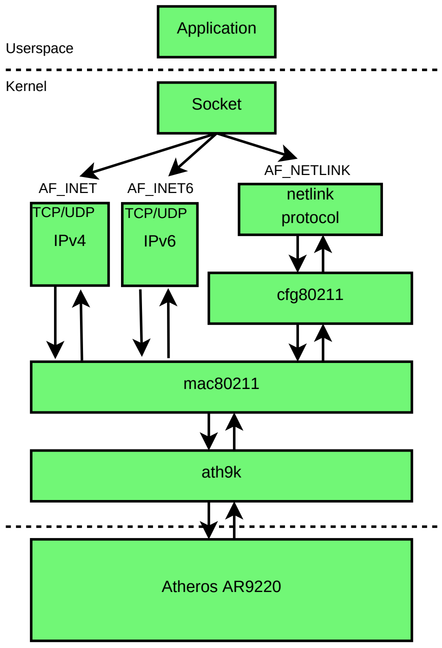
The above depicted structure allows for abstractions at the different layers, thereby ensuring the independent operation of the layers, that is, the higher layers do not require knowledge of the physical layer and vice versa. The current and next paragraph highlight the interactions between different layers while either transmitting or receiving a packet. The transmission of packets from userspace applications is performed by executing a system call, with at least the socket file descriptor and the data as parameters. The specific protocol layer, to which the socket is associated, will process the packet and perform a lookup of the network device (*struct net_ device*) through which the packet needs to be sent. One of the exported operations towards the network device is *ndo_ start_ xmit*, which calls the respective transmit function of the specific driver. The reception of packets is enabled by the upper layers to register callback functions for the specific packet types they can handle. The drivers signal the networking core that one or more packets are available, which are forwarded towards the respective handlers according to packet type. After the processing of the specific layer, the packet is placed in a buffer of the specific socket, where it waits until the userspace application performs a read operation on the socket.

Note that previous description of both transmission and reception is a simplified description of the path a packet traverses through the different networking layers to keep the discussion within certain boundaries. Amongst others, Qdisc, the NAPI interface, polling, backlog processing and softirq processing is intentionally left out of the discussion, since, although it is interesting information, it is out of scope of this section. Interesting is also that a packet is represented by a *struct sk_buff*, which is passed through the whole kernel networking subsystem. All passing is therefore done on a per packet base, or a list of packets.

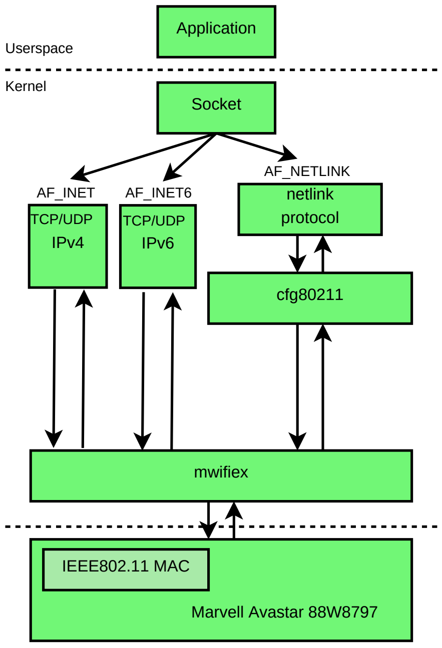
The above description covers all types of network interfaces up to a certain level. Ethernet drivers allocate a *struct net_device*, associate their personal operations with the network device and register the device. Most communication between the network device specific operators and the device drivers happens through the registered operation functions. There is a one to one mapping between the network device and the device driver. The network structure for wireless devices is significantly more complex and requires also the assistance of a number of additional kernel modules, such as cfg80211 and possibly mac80211.

A diagram of the network blocks, that are relevant to the wireless operation, is shown in Figure 6.6. Note that the representation is extremely simplified as, for example, the link between IPv4 and mac80211 represents the complete packet processing of the networking core. The depicted network structure represents both a SoftMAC (Figure 6.6(a)) and a FullMAC (Figure 6.6(b)) type of architecture. Concepts such as SoftMAC and FullMAC are common in the wireless kernel developer community and have a significant impact on the operation of the wireless driver. The SoftMAC refers to the mac80211 module, where all the MAC functionality of IEEE Std 802.11 is provided. From the viewpoint of the networking core, the mac80211 module is considered to be a driver and is also treated as such. The device specific drivers that make use of the SoftMAC, such as ath9k, register themselves at the mac80211 and allow it to perform the MAC specific operations. Device drivers that adhere to the FullMAC principle do not link to the mac80211, since the device specific IEEE Std 802.11 MAC implementation is often provided in the firmware or even in the hardware, thereby making the mac80211 functionality redundant. In the remainder of this section the usage of the mac80211 module is assumed as the operational functionality of the IEEE Std 802.11 MAC.

Irrespectively of which concept, SoftMAC or FullMAC, is employed, a link towards the cfg80211 module is required, either from the driver itself or from the mac80211 module. The cfg80211 module provides an interface between userspace and the IEEE Std 802.11 MAC, besides the commonly employed socket interface of the networking core. The link towards userspace is implemented by means of the netlink protocol, which is a mechanism to transfer information between kernel and userspace [206]. The netlink protocol allows userspace applications to send data packets, contain-



(a) *Softmac kernel structure*



(b) *Fullmac kernel structure*

Figure 6.6: *SoftMAC and fullMAC concepts*

ing configuration information which needs to be formatted in a specific manner, towards kernel subsystems through the AF_NETLINK socket. Kernel modules can send messages to userspace in an asynchronous manner by means of the internal kernel API. Kernel subsystems can register themselves as listeners for a specific netlink family. The cfg80211 module is registered as such a listener and is able to parse the messages that contain requests or commands for the wireless subsystem. The interface towards the MAC is provided by a *struct cfg80211_ registered_ device*, which will be discussed later in detail. The cfg80211 module has a wide range of responsibilities. First, it provides the notification of MAC Layer Management Entity actions, such as authentication, deauthentication, etc., as well as the passing of received management packets by means of the netlink protocol. Second, cfg80211 provides support for regulatory infrastructure, that is, it specifies the boundaries that should be adhered to according to the country regulations. Third, the scanning of the wireless medium is a task performed by the MAC, however, the list of found Basic Service Sets (BSSs) is maintained by the cfg80211 and can also be exported to userspace. Fourth, cfg80211 maintains the state information for the Station Management Entity (SME). The decision to proceed to the next step in the association phase is considered in this module, but it requires all its information from the MAC. Other tasks include the parsing and composing of the netlink messages, verification of channel information, providing an interface for ethtool, parsing of radiotap headers of injected packets, etc. Note that the discussed functionality applies solely to either stations or Access Points; devices operating in ad hoc or Mesh mode are not discussed.

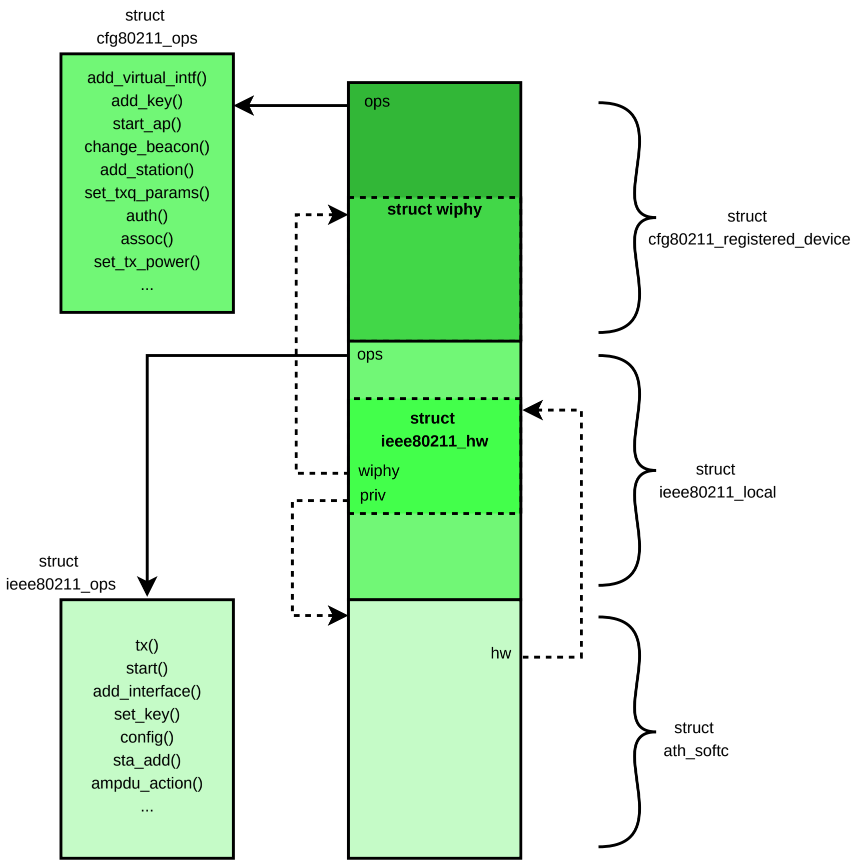


Figure 6.7: *Physical interface representation*

The mac80211 module is considered as a device driver from the viewpoint of both the networking core and the cfg80211 module. The complexity is substantiated by the required number of data structures for a single wireless device, which is in the order of several dozens, of which eight are large core structures. Most of these data structures are used in the mac80211 and cfg80211 modules. The set of data structures is required for the complete IEEE Std 802.11 MAC operation, in different operating modes and states.

A physical wireless device is represented by a subset of these data structures, depicted in Figure 6.7, which contain mostly hardware related information, a shared data structure with the wireless driver and a structure the cfg80211 module operators can manipulate. The wireless driver initiates the construction of the relevant structures that represent a physical wireless device. When assuming that the network card is connected to the CPU through a PCI type of bus (PCI, Mini PCI or PCI Express), the driver passes a set of operators and information to the PCI subsystem while registering itself as PCI driver. One of these operators, a probe function, is called when a matching PCI id has been found by the PCI subsystem. As a result of calling this probe function, the data structure of Figure 6.7 is created, representing the physical wireless device. During the operation, the wireless driver registers itself at the mac80211, providing possible operators of the wireless device driver to the mac80211 in the form of *struct ieee80211_ ops*. Similarly, the mac80211 registers itself to the cfg80211, by creating a *struct cfg80211_ registered_ device*, while at the same time providing a set of operators in the form of *struct cfg80211_ ops*.

As depicted in Figure 6.8, for each physical wireless device, a list of virtual interfaces is maintained, of which each is represented by another subset of data structures, amongst which a *struct net_ device*. Therefore, a wireless device can be represented by multiple network devices, of which each can function in a different operational mode. In order not to elaborate too much on the specific implementation details, the individual data structures are not discussed in detail. A general description of the operation is provided in the following paragraphs.

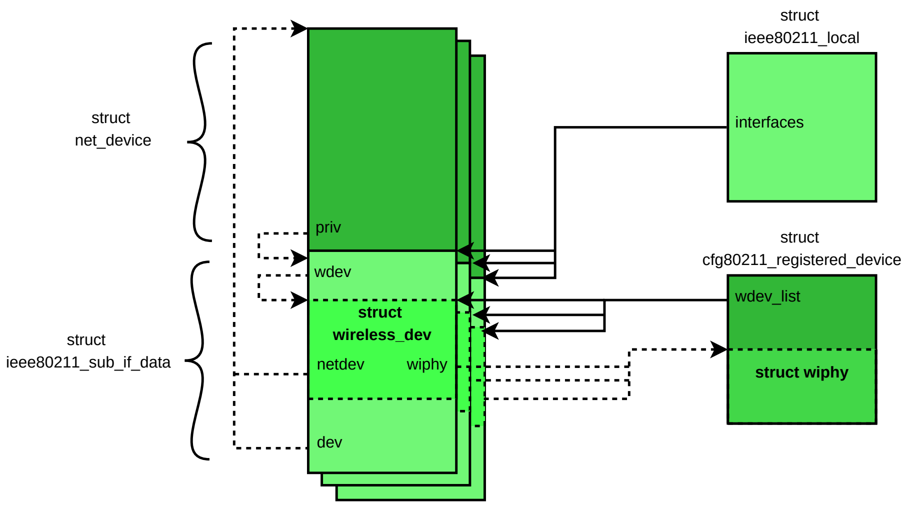


Figure 6.8: *Virtual interface representation*

The mac80211 provides the IEEE Std 802.11 MAC functionality for station, AP, mesh and ad hoc (IBSS) mode. Besides those operating modes also vlans, monitor mode, P2P and WDS mode are supported amongst others. One of the main responsibilities of the mac80211 is evidently the transmission and reception of packets to and from the device driver. During packet transmission and reception, a series of packet handlers are invoked to manipulate the packet. The transmission path for example checks for interfaces in station mode whether dynamic power saving (U-APSD) is enabled. If it is enabled and the specific QoS class (WMM, also known as WME, a subset of IEEE Std 802.11e), to which the packet is classified to, is configured to support U-APSD, the packet is passed on to the next handler. For any other case, if the interface is operating in station mode and is currently in power saving mode, the power saving mode is disabled when the corresponding queued work structure is processed. Other handlers address whether or not the station is associated or the Access Point has associated stations, select the appropriate encryption key, allow the rate control algorithm to determine the most optimal rates, allocate sequence numbers, fragment the packets if needed and encrypt the fragmented packets. Each of the handlers has the option to drop the packet in case certain condition are not fulfilled, such as when the device is operating in AP mode, while no associated stations are known, the packet is dropped.

The reception of a packet is structured in a similar manner, first duplicate packets are dropped, check whether the device is associated, notify the MLME section of mac80211 of the arrival of a new packet, such that its connection monitor, whose operation is similar to a watchdog, can be reset. Other handlers drop nullfunc messages, whose sole purpose is to notify about power saving mode, after checking the power management bit. The packets are consequently decrypted and defragmented, after which each handler provides specific handling for certain packet types. For example, packets that are aggregated in an A-MSDU packet are processed and sent to the higher layers one by one. Similarly, management packets are sent to the respective management handler, for each of the possible operating modes. In the case where the interface is operating in AP mode, no specific handler functions are provided, unlike modes such as station, IBSS or mesh. In such case, the management packets are, like with all data packets or management packets that have not been handled, sent to the monitor interface when available, which forwards them to any userspace application listening to a monitor interface.

Other than the transmission and reception of packets, the mac80211 has a wide range of responsibilities. As already briefly mentioned, the MLME packet handling of interfaces in station, IBSS or mesh mode is supported by the mac80211, that is, the packets are parsed, the appropriate timeouts are enforced and the necessary actions are taken. In the case of station mode, notifications are sent from the mac80211 towards the MLME section of the cfg80211 module, which is mainly intended to notify the respective listeners of the changes, such as the SME section of cfg80211. The latter provides the station state transition logic, which passes commands back to the mac80211. The AP mode management packet handling is not provided by the mac80211, or elsewhere in the kernel. For this, a userspace application, such as hostapd is required, which will be discussed in a later section.

The mac80211 is also responsible for the creation and removal of virtual interfaces and opens them on command of cfg80211. When opening a virtual interface, the device driver is activated if this has not been done already while opening other virtual interfaces on the same hardware. For a correct network operation, the mac80211 maintains a record of associated stations, in AP, IBSS or mesh mode, and uses the same structure for a client to maintain information regarding its AP. Other tasks include performing scan operations, rate control and encryption if the hardware does not support it, block negotiation and evidently providing an interface to cfg80211. Note that the accepting of packets from the higher layers is not performed through the cfg80211 interface, but through the interface with *struct net_ device* on a virtual interface base.

The mac80211 is a very flexible module and a considerable number of parameters and settings are determined by the functionality the device driver and the hardware itself provide. For example, if a device in AP mode encounters a broadcast packet in its transmission queue, while some of its stations are in power save mode, it buffers the packets in one of its queues. Such case is under the assumption that the hardware does not support power save buffering. If the hardware supports the functionality, the task is left to the hardware to execute. Likewise, the mac80211 is capable of performing some of the encryption and decryption algorithms in software, if the hardware does not support those encryption methods.

Device drivers, such as ath9k, register themselves at the mac80211, as already mentioned before. The operation of the ath9k driver is not discussed in this section, since its operation relies heavily upon the hardware it needs to control. In a later section, when discussing the modifications made to enable a reliable transmission scheme, an elaboration regarding its operation and its corresponding hardware is provided.

6.5 Real-Time Systems

A real-time system is a system whose correctness depends not only on the logical results of the computations, but also on the time at which the results are produced [44] [207]. In other words, the result might be logically correct, but if it is not produced on time, the result is useless or even dangerous. A distinction can be made between soft and hard real-time systems, based on the consequences of missing a deadline. Whereas missing a deadline can have catastrophic consequences in a hard real-time system, it would merely provide some inconvenience in a soft real-time system. An example of a hard real-time system would be paper cutting control system that verifies whether or not obstacles, such as hands, prevent the knife from cutting the paper. Should such system come too late to the decision that the cutter should be stopped, that could have severe consequences. An example of a soft real-time system could be a video conferencing system, where no buffering is used. In case one frame of the video stream arrives too late, then either one frame is skipped, or it is shown anyway. The user could detect a short blimp, which could cause some annoyance, but he will not experience any severe or lasting dammage because of it.

Although the timeliness is of utmost importance for an RTOS, the system is not required to have a fast response time [208]. Important is that deadlines are met, irrespective of the size of the time frame, which is dependent on the type of environment. The predictability of the system is the most important property, that is, the system should provide guarantees that all timing constraints are met. Unfortunately, several methods and features that are introduced to improve the efficiency and average performance of a system, such as Direct Memory Access (DMA) and memory cache, also form a source of nondeterminism [207]. The goal of DMA is to take over the control of the I/O tranfer from the CPU and thereby relieving the CPU. However, the I/O device and CPU share the same bus towards the memory. As a result, the CPU needs to be blocked while the DMA transfer is ongoing. This is often by means of cycle stealing, where the DMA steals CPU memory cycles in order to execute a transfer. Since the program execution at the CPU is running in parallel with the DMA transfer, it could happen that the CPU requires the memory bus, which is at that moment occupied by the DMA transfer. Therefore, such an improvement causes a certain indetermination in the system, since it can not be predeicted how many times or when the CPU will have to wait for the completion of a DMA transfer [207]. The cache memory is also introduced to improve the overall system performance, but like DMA, it introduces a certain degree of indeterminism. The system is responding fast when the requested data can be found in the cache, but when it is not found, an extra operation, that is, the retrrieval of the data from memory, needs to be executed in order to obtain the requested data. Even though the operation is the same from the point of view of the program, the duration of the operation might be different. In order to take this into account, one would need to assume that a cache miss occurs for every lookup, which is why it is more efficient to disable the cache for such real-time systems [207]. Likewise, while interrupts improve the overall system response time, since interrupt handlers are usually executed with the highest priority, they introduce nondeterminism due to their asynchronous character. Even real-time processes are subjected to the interrupts of possibly irrelevant devices. Therefore, the design of an RTOS needs to take into account amongst others scheduling algorithms, memory allocation, interrupt handling etc.

It is common knowledge that the Linux OS (GNU/Linux OS) is not a hard Real-Time OS (RTOS). It is said by some people that Linux does exhibit soft real-time properties [209]. However, the system does not provide real-time guarantees, even while stringent timing requirements can be achieved. Some issues that introduce nondeterminism are mentioned in [208], such as non-preemptable critical sections with non-predictable duration during which interrupts are possibly disabled, such as in the printk call. The Linux system is designed as a general purpose OS (GPOS), which attempts to schedule all processes in a fair manner (Section 6.4.1).

Notwithstanding the GPOS character of Linux, it became popular to modify the Linux system into a real-time OS. Two different methodologies are employed to ensure real-time operation. The first method modifies the Linux kernel in order to introduce real-time characteristics, such as the CONFIG_PREEMPT_RT patch project [200].

The CONFIG_PREEMPT_RT patch is not intended to provide hard real-time guarantees towards the users. Instead, its goal is to improve the user's experience [210]. Since the interrupt handlers and interrupt disabling during critical sections results in unpredictable latencies, a major remodeling of the interrupt handling and locking is performed. Most interrupt handlers, both hardware IRQ handlers as softirqs (including tasklets), are transformed in kernel threads. A small interrupt handler catches and acknowledges the interrupt and schedules the respective interrupt thread. Since all interrupt handlers run in process context instead of interrupt context, priorities can be assigned to each individual thread. One of the exceptions is the hrtimer interrupt, which is directly handled by the hrtimer.

Since most interrupt handlers are running in process context, there is also no need to disable interrupts in a significant number of critical sections. The traditional *spin_lock_irqsave* and all its variants are therefore modified to use the new rt-mutex, which supports priority inheritance, while only some critical sections use the *raw_spin_lock_irqsave*, which implements the original behavior of *spin_lock_irqsave*. This information is based on patch-3.2.44-rt64.

The second method to employ the Linux system in a real-time environment is to introduce a second micro real-time kernel, such as in the Xenomai project [211]. This discussion targets *Xenomai 2* and adds a little side note concerning the newer *Xenomai 3*. The Xenomai real-time system is available as a patch to the Linux kernel and provides a small co-kernel running side-by-side with Linux [210]. In order to maintain close control over the interrupt processing, a layer, an interrupt pipeline, is introduced between the hardware and both Xenomai and Linux. This interrupt pipeline, also called I-Pipe, is derived from the interrupt pipeline contained within the Adeos project[212], whose purpose was to enable the sharing of hardware between multiple operating systems [213]. The interrupt pipeline allows different systems to be connected by a software pipeline by organizing them into domains. For each domain a virtual interrupt mask is available, which allows the domain to stall interrupts when for example entering a critical section. The Xenomai core is running at the highest domain priority, which is the first domain to receive interrupt signals. Xenomai has the possibility to accept, ignore, discard or terminate interrupts [213]. When accepting the interrupt, the interrupt can still be passed on to the next domain. Discarding an interrupt makes the I-pipe to skip the specific domain and offer the interrupt to the following domain, while terminating an interrupt means the interrupt will not be presented to any of the following domains.

In order to facilitate the porting of different RTOS applications towards Xenomai, different *skins* are implemented, which provide an API for the specific RTOS applications. Each skin is contained within a single loadable Linux kernel module and interacts with the Xenomai nucleus, where all basic building blocks are maintained. One of the skins, Real-Time Driver Model (RTDM) is to be employed specifically by real-time drivers. Real-time applications are scheduled by the Xenomai real-time scheduler, which is a fixed-priority preemptive scheduler. They start as regular Linux processes and declare themselves as real-time applications after initialization by calling one of the calls defined by the specific *skin*. This places the process in primary mode, that is, the process is now being scheduled by the Xenomai scheduler instead of the Linux scheduler. When accessing any Linux specific service, such as a system call, they are placed back into secondary mode, where the Linux kernel performs the scheduling[214].

Besides all previously discussed features, Xenomai also provides its own synchronization methods to allow thread blocking with support for timeouts, priority inheritance and priority-based or FIFO queuing order. A memory allocator with predictable latencies, a generic interrupt handler and timer management are also provided by Xenomai.

In *Xenomai 3*, support has been added for native Linux configurations besides the dual kernel approach. The *Xenomai 2* kernel is redesigned into the *Cobalt core*, where only the RTDM skin remains available for kernel processes and a *Copperplate*, which provides the interface between the real-time application and the real-time kernel. In case of the native Linux configuration, where a Linux patched with the PREEMPT_RT patch is used to be able to meet stricter timing requirements, the *Cobalt core* is not used anymore. Only a standalone implementation of the RTDM specification is required for the real-time drivers, while the real-time application interfaces to Linux by means of the *Copperplate*.

While the two previously discussed real-time approaches are Linux based, the following RTOS to be discussed is developed independently of any other OS. eCos [215] is an RTOS developed to provide a cost-effective, high-quality embedded software solution. This discussion considers the operating system only superficially. A more profound analysis can be found in [216], which proved to be the main source for this discussion.

Since one of the goals of eCos was to provide a minimum resource footprint, the OS was designed to be highly configurable. All functionality is contained within a framework of reusable software components, each of them configurable. Embedded software often includes generic support functionality which is not required for the application to run. This extra code makes the software unnecessarily more complex. eCos allows to exclude unneeded functionality from the build, allowing the build to scale from a few hundred bytes up to hundreds of kilobytes, depending on the included functionality. The software component control is performed at compile time, ensuring best results in terms of code size.

The core components of eCos include the kernel, the hardware abstraction layer (HAL) and device drivers. The kernel considers interrupt handling, support for threads, two different real-time schedulers and a timer mechanism. Note that device drivers are not part of the kernel to facilitate the exclusion of unnecessary functionality. The interrupt handling scheme is designed to reduce the interrupt latency and be deterministic; a split interrupt handling scheme is employed. The interrupt is directly serviced by the Interrupt Service Routine (ISR), which performs only the most critical tasks, while further interrupt processing is deferred to the Deferred Service Routine (DSR). An ISR has the highest priority and a DSR has a higher priority than threads, however, if a thread has locked the scheduler, the DSR is delayed until the thread unlocks it. Note the similarity with the Linux interrupt handling. In order to prevent recurring interrupts, eCos forces the ISR to mask the current interrupt, whereas the DSR needs to unmask the current interrupt after it has been serviced. The timer mechanism of eCos includes Counters, Clocks and Alarms. A Counter maintains a continuously increasing value that is driven by a source of ticks. Depending on the number of required Counters, either a simple list can be used or a more sophisticated structure for the Counters. A Clock is also a Counter, but has an associated resolution, which is driven by a regular source of ticks. The default system clock is the *Real-Time Clock* (RTC). An Alarm is associated to a Counter and signals an event based upon the value of the Counter. This handler needs to comply to the requirements that DSRs need to comply to.

6.6 Conclusion

The employment of IEEE Std 802.11 COTS hardware in a non-standardized manner has been and still is a popular research topic. Thanks to the low cost and the world wide integration of the devices, they are considered interesting to work with. Unfortunately, the standard seems to lack some features or causes a suboptimal performance for certain use cases. The discussed related work included WiFi deployments in rural areas, guaranteed QoS provisioning, power efficiency and the more academic interests where research is being performed to extend the functionality as far as possible beyond the specifications of the standard.

The long-distance deployment in rural areas encountered issues regarding the default timings of the standard, which have an impact on carrier sensing and acknowledgements. Most of these works therefore proposed to use a TDMA scheme on top of the COTS hardware by means of the MadWiFi driver. The transmission of real-time voice and video streaming requires a guaranteed QoS from the network. Some works ensure this by implementing a TDMA MAC in the MadWiFi driver. The MAC employs the Linux software timer, since it is said to be more reliable than the hardware timer. Other works manipulate the IFS or AIFS timing and the minimum and maximum backoff contention window, such that VoIP packets are allowed to access the medium earlier than any other type of packets. In order to prevent collisions between VoIP transmissions, they are distributed in time. The more academic research made advantage of the open source character of the Linux kernel and its drivers in order to pursue a wide variety of goals. Some of these goals are purely academic, such as the modularization of the HT and MLME functionality, while others are more targeted towards a specific goal. Either the MadWiFi or more recent drivers, including the mac80211 Linux kernel module, are employed to achieve these goals. Some works consider the power efficiency of the network by eliminating idle listening. The considered work achieved this by means of a TDMA MAC protocol.

Since the employment of COTS hardware in a non-standardized manner is only made possible thanks to the open source character of the Linux kernel, the related work also includes some works regarding the performance evaluation of certain kernel subsystems.

As the goal in the following chapter is the transmission control of COTS hardware, a clear understanding of the basics of the IEEE Std 802.11 standard is provided in this chapter regarding functionality which could influence the transmission timings. Most of the functionality is discussed within the context of a BSS. One of the most prominent factors which could influence the transmission timing is the DCF, where the different interframe spaces are discussed, such as DIFS, SIFS, AIFS, etc. The minimum and maximum contention window, RTS and CTS, and retry counters are also considered in the discussion. Since the hardware and Linux kernel have built-in support, which are by default enabled, for the QoS provisions defined in the standard, the HCF, EDCA, TXOP and the different access categories are also considered in this chapter. Other discussed topics include the Block Ack method and its negotiation, which have a significant impact on the performance of the network, as well as both service and protocol aggregation, and power save mode. One last essential piece of information about the standard is delivered in this chapter while discussing the association procedure.

Since the open source character is paramount for the success of the non-standardized operation of IEEE Std 802.11 COTS hardware, some of the relevant internals of the kernel are discussed in this chapter. The depicted functionality includes the process scheduling, interrupt processing, software timers and evidently the networking subsystem, where the focus of the discussion was on the SoftMAC method. The specific driver internals are not discussed, since this is implementation and hardware dependent, while an attempt is made to maintain this discussion as general as possible.

The last section superficially discusses two methods to either approach the real-time functionality or exhibition of actual real-time functionality of the Linux kernel. As a comparison, a dedicated RTOS is also discussed. The real-time operating systems are relevant since the scheduling of the transmission process is very critical. Therefore, having an OS where deadline guarantees are provided could prove to be beneficial to the performance.

CHAPTER 7

Transmission control on a IEEE Std 802.11n compliant Network Interface Card

7.1 Introduction

This Chapter depicts the analysis of timer sources in the Linux kernel in Section 7.3 and the implementation details of how to implement a reliable transmission on COTS hardware in Section 7.4 The ever increasing bandwidth, low cost and relative ease of deployment make IEEE Std 802.11 compliant hardware an interesting means to realize setups for real-time multimedia applications. Such application can constitute an audio-visual communication and data collaboration in a network conference environment, such as in [217]. The proposed method comprises a central server which runs an audio/video (AV) server, alongside a data server. The local or remote client run an AV client and/or a data client. The AV server-client connection allows for a real-time communication means between the central server and each of the clients, while the data client-server connection allows the exchange of additional information, such as presentation slides. The scenario allows for bidirectional communication between the server and each of the clients. Note that the transferred AV stream contains time sensitive information whose arrival is subjected to jitter and delay boundaries. Although the high throughput specification of the IEEE Std 802.11 provides sufficient capacity to implement such scheme, the said rate improvement is an insufficient measure when not accommodated with QoS guarantees. Whereas the standard does provide control of the QoS parameters, it does not enforce guarantees. Neither does IEEE Std. 802.11e provide any QoS guarantees, where high priority traffic is concerned. The improvement ensures that the packet has a higher probablity of being sent first, but does not provide any guarantees.

Each data transmission should be precisely timed in order to guarantee the timeliness, even in environments where a large number of electronic devices are present, which are potential interferers, sending probe requests to the Access Point (AP), trying to perform an association. While manufacturing a chip that is able to operate in TDMA access mode would prove to be a very efficient solution, it would require an extensive outlet due to the high development cost. The operation of the IEEE Std 802.11 compliant hardware in TDMA access mode would provide a solution for this issue. The open source character of the Linux kernel,the SoftMAC principle and the configuration interface of the COTS hardware, made available to the device driver, allow the adaptation of the Linux kernel wireless subsystem such that TDMA access mode is made possible. The previous chapter indicated already the large number of works that has focused on devising a method where TDMA access mode can be used in combination with commodity WiFi cards. Most of these works are based on the older IEEE Std 802.11b/g specification, thereby not taking into account possible aggregation or Block Ack mechanisms. Moreover, due to the older hardware, the higher speeds of for example IEEE Std 802.11n are not possible. Whereas some works provide a precise description how to circumvent the CSMA/CA operation of the hardware, other works maintain rather superfluous as regards their method. Although most of the related work does not elaborate on it, additional care should be taken in the avoidance of inadvertent receptions, such as probe requests from regular devices outside of the network, since they could jeopardize the timing schedule of the transmissions.

The success of the TDMA algorithm depends on a number of critical factors, which determine an upper bound on the performance. One of those factors is rarely discussed in the related work, that is, the precision of the timer source. The TDMA algorithm requires a precise timer source in order to determine the respective slot boundaries. Some works clarify the reasons for choosing the specific timer source, however, neither an extensive performance analysis nor an analysis of the timer code has been performed to compare the available timer sources. When considering how precise such timer source really should be, the maximum available throughput of the employed technology needs to be taken into account. This chapter discussed the IEEE Std. 802.11n standard, where the maximum rate, when using a single spatial stream, a 40 MHz channel, a short Guard Interval and MCS 7, would be 150 Mbps. This results in the transmission of a single bit around every 7 ns and the transmission of a single byte around every 56 ns. That would signify that around every 80 µs a packet needs to be transmitted in order to achieve this throughput. The precision of a suitable timer would therefore need to be lie within the µs range or lower, to prevent too much jitter on the packet transmissions and as a result of the jitter, the ineffective use of the available time.

This chapter studies in the first place the stability and performance of available timer sources. This is performed both by analyzing the timer code and by means of a performance analysis. Different kernel versions have been compared as regards the timer performance. The influence of preemptiveness, HZ configuration (which is a kernel compile-time constant that specifies the time interval between subsequent system ticks [218]) is verified by completing the analysis for different configurations of the respective parameters. Since the timer precision is a critical factor to the success of the TDMA algorithm, a Linux kernel enhanced with the RT-patch, a Linux kernel patched with the Xenomai real-time extension as well as a real-time Operating System (RTOS), such as eCos, is used in the comparison. All comparisons are performed in a regular situation, as well as in a high load situation, which can comprise either timers running in parallel or high throughput network operation. Since the hardware platform determines the performance and precision of the clock system, two different hardware platforms are considered for the performance tests.

In the second place, the required adaptations to the Linux kernel wireless subsystem, to enable a reliable TDMA access mode, is discussed. The chapter elaborates on achieving a slotted packet transmission with a resolution of a single packet every 256 µs, where the jitter should be constrained to 10 µs. The focus of this chapter lies on the accuracy of the scheduled transmission, not on the actual TDMA scheme. The manipulation of the Linux kernel wireless subsystem, such that TDMA operation is made possible, poses several challenges, for which a solution is proposed, even when working in a noisy environment.

The work in this chapter is based on different kernel versions, ranging from 2.6.38 to 3.13.9. The wireless Mini PCI network card is based on Atheros AR9220, an IEEE Std 802.11n compliant chipset. Therefore, most, if not all, discussions in this chapter refer to the IEEE Std. 802.11n standard, especially since this standard is being used as the reference for the Linux kernel code and NIC hardware architecture. The hardware development platforms that were employed include a regular PC architecture with AMD Sempron$^{\text{TM}}$ 2400+, an SBC architecture ALIX3D3 500 MHz with AMD Geode LX800, and an IGEPv2 board with a 1 GHz ARM CortexA8 processor.

The next section identifies the possible timer sources and exposes the methodology to analyze the performance of the timers. Section 7.3 discusses the obtained performance results of the different timer sources. Several concepts that should be considered when making the required adjustments are explained in Section 7.4. The implementation for the controlled slotted transmission is discussed in Section 7.5. The last section concludes this chapter.

7.2 Timer source analysis

It is commonly known that the Linux kernel does not provide real-time support, hence the several real-time Linux OS versions or kernel extensions that should provide real-time behavior to the kernel. This and the following section analyzes whether the performance of any of the available timer sources is sufficient to support a precisely controlled transmission scheme. This section focuses on the identification of timer sources, what can be considered as a timer source, and the measurement methodology. The discussed timer sources are timers, available at kernel level, since the timer would need to trigger a routine in one of the kernel drivers. The following section discusses the measurement results and deliberates on the comparison between the timer performances of both kernels with different parameters as well as real-time systems.

7.2.1 Timer source identification

The number and type of timer sources are dependent on the hardware development platform. In order to maintain a certain abstraction towards the hardware and remain hardware independent, that is, no additional specific hardware is required for the said timer source, only Linux software timers are taken under consideration. An exception to this rule is the timer source available within the wireless Network Interface Card (NIC), which is an essential component of the wireless system. Note that even though solely considering timers that are usually available on contemporary platforms, the type and resolution of the timers can change when migrating to a different hardware platform.

The identified timer sources that are considered as potential trigger inputs for the slotted transmission consist of the General Purpose timers of AR9220, which also includes the Software Beacon Alert timer, the Linux timer wheel, Linux hrtimer and a proprietary high priority timer. The first timer is embedded in the Atheros hardware, while the last three timers are Linux software timers.

The Software Beacon Alert (SWBA, later also referred to as bcntimer) is one of the eight timers that constitute the General Purpose timers of the AR9220 chipset. The timer is usually used to notify the station about an immanent elapse of the beacon interval. The notification is expressed as an interrupt on the PCI bus, which needs to be handled by the device driver which communicates to the hardware device. Stations that need to send beacons, such as an Access Point (AP), are required to prepare a beacon packet and place it in the hardware transmission queue. According to [169] and [172], the timer is performing well under idle conditions, however, degrades fast with an increasing network flow.

In order to verify the performance of the SWBA, the interrupt handler, which schedules the beacon handler tasklet upon noticing the SWBA interrupt bit, is modified, such that upon reception of the interrupt, the *timer performance monitoring function* is called instead, which is defined in Section 7.2.3. Since the SWBA timer is part of the same hardware core as the other timers, analyzing the performance of the said timer is sufficient. The performance of this timer, compared to other timers that are being discussed here, can be found in Section 7.3.

The Linux kernel timer wheel is a basic software structure in the kernel (See Section 6.4.3), often used to specify timeout handlers. Due to the low resolution and dependency on another software timer, this timer is not considered in the performance analysis.

The hrtimer on the other hand is a high resolution Linux software timer, where all timers are organized in a Red-Black Tree per processor (See Section 6.4.3). Upon expiry of the scheduled hrtimer, the hrtimer_interrupt function is called, which calls the handler function of the timer. During initialization of the performance measurement timer, its handler function was configured as the *timer performance monitoring function*, which resides in the kernel module responsible for the performance measurement.

From a code analysis, it is clear that most time critical operations are done within interrupt context, either within an interrupt handler or a softirq. Since the execution within those contexts takes priority over almost anything else, the triggering of the timer handler is meant to be very precise. As regards kernel configuration, not many configuration options, such as HZ value, preemptiveness and dynamic ticks, are expected to exhibit any significant difference in timer performance. The HZ value defines the tick interval, that is, the periodical timer interval which updates the tick count, performs process accounting, etc. Its influence to the hrtimer is limited to the scheduling of a single timer at a maximum timer interval of 1 ms (HZ = 1000) The type of preemption is a significant factor within the process scheduling context, however, in interrupt context this parameter should not manifest any influence on the performance. Likewise is the expected influence of the dynamic tick configuration on the performance of the hrtimer nondescript. The tick counter can be considered as just one of the timers that make use of the hrtimer structure. Moreover, the dynamic tick configuration is only applicable when the CPU enters idle mode and the idle time is longer than the configured tick rate. The performance measurements have a time interval which is generally smaller than the tick rate, thereby not allowing the use of dynamic ticks.

Since previous statements are deduced from code analysis, a performance measurement has been done with a variation of all parameters to verify former statements. The HZ parameter takes values of either 100 or 1000 in the performance measurements, the preemptiveness is either a preemptible kernel (Preemptible Kernel, Low-Latency Desktop (CONFIG_PREEMPT)) or a non-preemptible kernel (No Forced Preemption, Server (CONFIG_PREEMPT_NONE)), and the dynamic ticks are either enabled or disabled. Preemption of a process signifies that the scheduler is stopping the process in favor of the actiovation of another process, which might have a higher priority, or has not been running for a long time, etc. After that process has been executed, or preempted, the scheduler can decide whether or not to resume with the preempted process. The Linux kernel is also one of the processes that needs to run, alongside all other processes. In the past, the kernel was not preembtible and it would need to relinquish control, before the scheduler could activate another process instead. A fully preemptible kernel allows low priority kernel processes to be preempted, even when they did not relinquish control of the CPU, provided that it is safe to do so. The goal of this configuration option is to reduce the kernel latency.

Note that only a single Red-Black Tree represents the hrtimer structure per CPU and only a single clockevent is attached to the hrtimer structure. Therefore, in a system with only a single CPU, all high precision timers are scheduled on the same timer structure. It is therefore imaginable that the deadlines of two timers are thus close to each other that the time to start the hardware timer is too short in order for the timer to fire in time. The Linux kernel solves this issue by reiterating over the elapsed timers with a new base timestamp to which the timer deadline is compared to. If the base timestamp is past the timer deadline, the timer is executed. In order to increase the efficiency of the hrtimer interrupt processing, a certain soft timer deadline is introduced, which indicates at which point in time before the actual deadline it is acceptable to trigger the execution of the timer handler.

Since the goal is to have a precise time reference, which does not execute too soon, nor too late, a proprietary timer structure is designed. Since the Linux kernel only creates sufficient clockevents for its own use, several hardware timers remain unused. The idea is to employ one of the free hardware timers and make them a dedicated timer for the proprietary high priority timer structure. Such timer functionality, called hptimer, is at its core based upon the hrtimer functionality, but otherwise differs significantly.

Upon boot time, the hptimer functionality makes an attempt to allocate the best possible clockevent, different from the clockevent already allocated to the hrtimer. When a hptimer is created, it is assumed that the timer is a periodic timer, unless otherwise stated. Unlike the hrtimer, the hptimer functionality is responsible of the repeated rescheduling of the timer. The timer storage structure could be a Red-Black Tree or even a simple list, since the expectation is that not a huge amount of timers is going to use this structure. The targeted timers are timers with the highest priority for which it is of the utmost importance that they provide a precise timer interrupt.

Upon the creation of such hptimer, the hptimer functionality checks whether the timer is divergent with all other timers that are already stored, based on its periodicity and start time. If this condition is not being fulfilled, the timer will try again at a different CPU, until either a fit CPU has been found or not found at all. This procedure will make sure that no timer expiry will collide with another timer. Moreover, the mechanism ensures there is sufficient time to program the clockevent for firing the next deadline. The actual timer interrupt routine is kept as short as possible and at every interrupt a pointer is made to the next deadline, to minimize the time required to update the hardware timer when required. A tasklet will update the timer structure in the background.

7.2.2 Real-Time Operating Systems

Due to the required precision of the timers, it is imperative to make a comparison with kernels that are enhanced with real-time patches and a real-time OS. Note that interrupts are not always welcomed by an RTOS, they disrupt the regular, that is, deterministic, operation of the system. However, timer interrupts usually are an exception to this rule. Performance measurements have been performed with two kernel patches and a single RTOS.

The most straightforward comparison is to enhance the Linux kernel with the RT-patch, which ensures that the kernel behaves in a more real-time manner, however, does not provide hard real-time properties. The performance measurement module is the same as in a regular kernel.

A kernel enhanced with the Xenomai real-time patch, ensures that a real-time microkernel is running underneath the Linux kernel. By means of the Adeos pipe interrupts are first addressed to the real-time kernel and afterwards to the Linux kernel, provided the real-time kernel decides to forward the interrupts. The real-time kernel also provides a software timer, which is used in this performance measurement. In order to employ such timer, a real-time kernel module needs to be created, which makes use of real-time callback routines. Beyond the real-time calls and the different timer, the performance measurement remains the same as with a regular kernel.

The last performance measurement is done with an eCos kernel. Since the eCos system is not compatible with Linux modules, a new measurement method is designed. A thread is started where a timer is configured to interrupt each interval. During the interrupt routine, the current time is stored. After a configurable period of time, the timer stops and prints out all stored measurement values, which can be stored on a Linux PC for further processing.

7.2.3 Timer analysis methodology

The capturing of timer interrupt trigger information is similar the procedure proposed in [188]. The use case is simpler for the timer analysis, since the related work needed to adjust several routines in the kernel code, whereas for the timer analysis, a kernel module can be created in which a timer is started and upon the timer interrupts the time is captured. However, in order to minimize the analysis overhead, a data structure is created, where the captured timestamps are stored and when the test has completed, this structure is transfered to user-space like in the related work.

In order to capture time information of the timer interrupt, a procedure is defined, called *the timer performance monitoring function*, which stores the current time, obtained by ktime_get, as raw data in the analysis data structure at every timer interrupt. When the requested number of measurements is reached the measurement stops and the buffer with the stored timing data is converted to a string buffer, which is used as a source for a debugfs file, which is available to user-space. The processing and analysis of the timer information is performed in user-space. The method is designed such that as little as possible measurement artifacts are possible by reducing the time in the interrupt routine to a bare minimum. Therefore, no string conversion functions or printk can be employed in the interrupt routine. The usage of printk during the whole measurement is inhibited since printk locks the cpu where it is running on and disables interrupts until it has finished its job. Since the timer is scheduled on a specific CPU, this could possibly prevent the system, both in SMP and single CPU system, from processing timer interrupts until printk has finished. The duration of such a printk operation depends on the length of the string and the speed of the CPU, but on an Alix board, a printk operation of normal length would take around 5 ms, that is, 5 ms during which no interrupts are being processed.

The goal of the analysis is not only to verify the timer performance in regular circumstances, but also to identify its weaknesses. The timer should be able to operate in a precise and stable manner, even when stressed. Since the timing critical sections of the software timers operate in interrupt context, their performance is largely defined by the interrupt handling speed and the efficiency of the timer deadline storage facility, which is inherent to the structure of the software timer. The interrupt handling is always an immediate action, unless some other process or interrupt has disabled interrupts, such as when entering some critical sections. As a consequence, an increase in system load, that is, a higher number of processes running in parallel, will not result in any deterioration of the timers, unless the higher load also leads to a significant increase in the number of interrupts or number of critical sections where interrupts are disabled. The Atheros timers, however, can also degrade due to excessive communication on the PCI bus.

In order to stress the interrupt handling system, not only measurements have been performed with only the timer under test, but also with other timers at different rates running in parallel, by using a userlevel program called cyclictest [219]. Placing such stress to the timer system allows the investigation of both the storing of multiple timers in the same software structure, and increasing the possibility of having two timer deadlines that collide at some point in time, and at the same time increasing the interrupt processing load.

In order to grasp the full picture of the timer performance, performance measurements have been performed on two different platforms with a different architecture. The first platform contains a x86 compatible processor, ALIX3D3, 500MHz AMD Geode LX800 cpu, miniPCI slot. The second platform is an IGEPv2 board with a 1 GHz ARM CortexA8 processor. The parallel timers, that ensure a stress on both the interrupt processing as well as on the timer structure, consist of ten timers, of which the timer interval increases with 100 µs for every timer, of the interfering timers which start with an interval of 100 µs. The timer interval of the interfering timers which start with an interval of 1000 µs stays fixed. Every measurement run collects timestamps from at least half a million measurements and the test is repeated ten times, after which a statistical analysis is performed on the results.

7.3 Timer analysis results

This section discusses the obtained measurement results regarding timer performance. The results are based on the raw timestamps, at which time the timer interrupt handlers were triggered. A post process analysis determines the inter-arrival times of the timer triggers and performs a statistical analysis on those data. In order to analyze the behavior of the timers in a thorough manner, several comparisons have been made, each showing a different aspect of the performance or eliminating a possible source of interference. First, the performance of the hrtimer is depicted for kernel 3.13.9. The different test scenarios include a variation on the preemptiveness, such as non-preemptive(server model) and preemptive(desktop model), and on the HZ values, such as 100 HZ and 1000 HZ. Second, a comparison is made between Linux kernel version 3.2 and 3.13.9 on the one hand and the hrtimer and the bc-ntimer (Atheros Software beacon alert (SWBA) timer) on the other hand. Third, the influence of dynticks and an active network transfer is investigated. Fourth, a proprietary timer construction, where the timer is assigned a dedicated hardware timer, is compared to the hrtimer. Last, the effect of employing one of the real-time patches and the eCos real-time OS is considered.

The results are represented by candlebar figures, which indicate the average inter-arrival time, marked by a dot, and the 95% confidence interval, which is illustrated by the top and bottom values of the bar. The different measurement scenarios regarding interfering timers are depicted at the x-axis, where nonoise is the case where no interfering timers are employed. The other cases relate to the interfering timer characteristics, where the first number indicates the requested starting interrupt interval and the second number indicates the real-time priority. In the case of eCos there is no priority for the interfering timers and therefore, only a single number identifies the interfering timers and the interval. Unless otherwise mentioned, the measurements are performed on the Alix platform while using the timestamp counter (tsc) as clocksource.

7.3.1 The hrtimer performance

The first analysis pertains to analyzing the hrtimer of the Linux kernel for kernel version 3.13.9. The Figures 7.1 and 7.2 depict the obtained measurement results for a 1000 µs and 250 µs interval respectively with different preemptiveness configurations and HZ configurations. The average value is in every case almost exactly the requested timer interval with a fixed deviation in the order of nanoseconds. It can be noticed that an increase in the number of timers running in parallel increases the jitter, that is, the deviation from the average value. The maximum deviation that can be observed now and then is limited to 40 µs for a 1000 µs interval and 45 µs for a 250 µs interval. Note that a deviation of more than 2 µs from the average value is observed only seven times in half a million samples. However, for both test runs a single outlier was detected while measuring with timers running in parallel at an interval of 100 µs, thereby blocking interrupts during three to four ms.

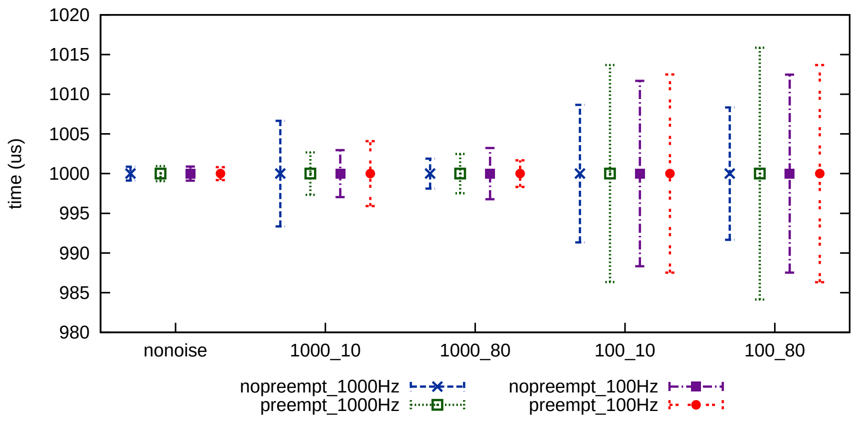


Figure 7.1: *3.13.9 hrtimer, variation of preemption and HZ, 1000 µs interval*

As can be noticed, there is no noticeable influence from the HZ configuration nor from the preemptiveness configuration. In one case the kernel with no preemption shows a better performance than a kernel with preemption, while in another case this might be reversed. The same goes for the HZ configuration. This result is to be expected, since the preemptiveness reflects itself on the process management level, while the HZ configuration is dependent on the hrtimer and can be considered as an extra timer running in parallel. On the other hand, the hrtimer is a low level timer, which is mostly influenced by the interrupt handling delay and the structure of the hrtimer that determines the speed with which the next timer deadline can be programmed. None of those things are influenced by the preemptiveness or the HZ configuration.

Another observation that can be made is that the priority of the interfering timers has no influence. It can be noticed that there is no direct relation between the timer priority and the jitter. This is to be expected, since the timer priority is the priority assigned to the user-space process, cyclictest, which processes timer events. Since the priority is related to the process context, it should not influence the basic behavior of the kernel timers on a regular Linux kernel, which operate in interrupt context.

Since from the previous paragraph could be deduced that neither the preemptiveness nor the HZ configuration deemed to have any influence on the results, the remainder of this section is limited to the results considering preemption (Desktop model) and a HZ value equal to 1000. In order to show the relation between the different

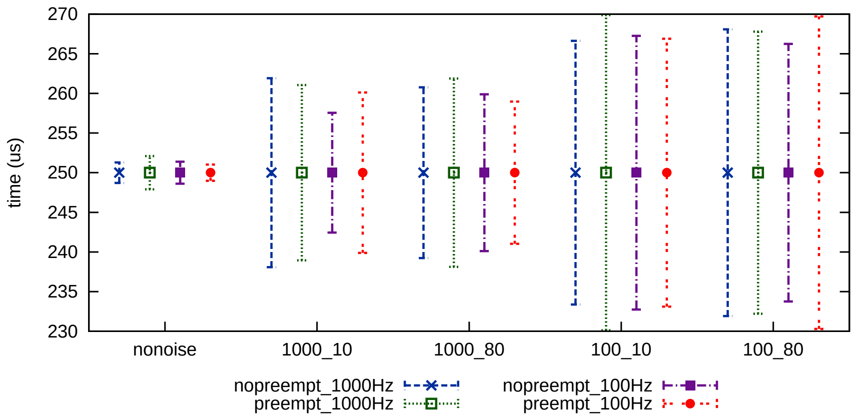


Figure 7.2: *3.13.9 hrtimer, variation of preemption and HZ, 250 µs interval*

results, the comparison will always include the hrtimer of kernel version 3.13.9, with preemption enabled (desktop model) and a HZ value of 1000.

7.3.2 The hrtimer in different kernels and the SWBA timer

The comparison of the SWBA timer (bcntimer) and the hrtimer is depicted in Figures 7.3 and 7.4 for a 1000 µs and 250 µs interval respectively. The measurements were performed at an earlier date and therefore also an older kernel version. The difference in kernel version has in general no effect on the bcntimer, which is embedded in the Atheros hardware. Modifications to the PCI core or the interrupt processing system of the Linux kernel might alter the behavior slightly, but this difference is insignificant. To demonstrate this statement, the figures include a comparison of the hrtimer with both kernel version 3.2 and kernel version 3.13.9.

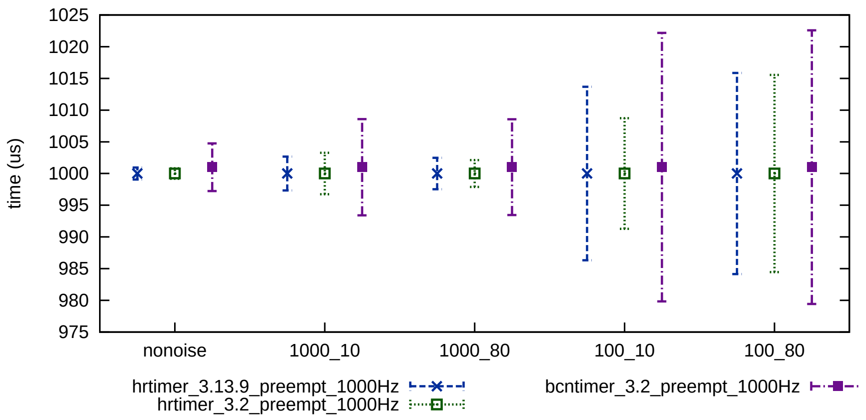


Figure 7.3: *SWBA timer vs 3.2 and 3.13.9 hrtimer, 1000 µs interval*

The figures indicate that the performance of the hrtimer of kernel 3.2 and kernel 3.13.9 is very similar, which can be expected since no structural changes occurred between the two kernel versions. The average inter-arrival time of the hrtimer depicts an excellent precision for both kernel versions. It is almost equal to the requested timer interval. The fixed deviation is in the order of merely nanoseconds. The SWBA timer on the other hand exhibits a less precise average timer interval where the fixed deviation towards the requested timer interval is found in the order of microseconds. This can be partially attributed to the coarser granularity of the bcntimer. From the figures can also be observed that the jitter is larger when using the bcntimer compared to the hrtimer. The absolute maximum jitter of the hrtimer is around 32 µs for the hrtimer at an interval of 1000 µs, and between 32 µs and 46 µs for the hrtimer at an interval of 250 µs (for both kernel versions 3.2 and 3.13.9). The bcntimer on the other hand exhibits an absolute maximum jitter around 77 µs and 79 µs for an interval of 1000 µs and 250 µs respectively. The maximum jitter value excludes the single case where during three to four ms no interrupt was handled and therefore resulted in an outlier. Those outliers occur once or twice during a complete test run, that is once every 5 million samples, especially when the interrupts are heavily pressed by running several high rate timers in parallel. The occurrence of these outliers is indifferent of which timer is being monitored and can be therefore attributed to the system and not some artifact of one of the timers.

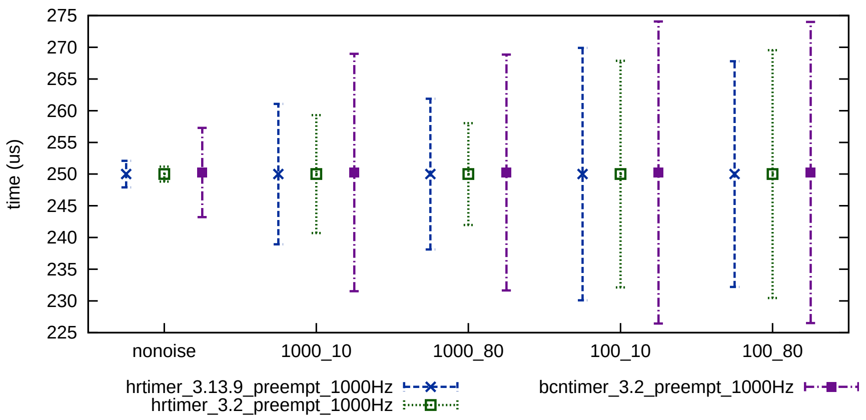


Figure 7.4: *SWBA timer vs 3.2 and 3.13.9 hrtimer, 250 µs interval*

Note also that the activation of extra timers running in parallel stresses the performance for both the hrtimer and bcntimer, even though their physiology is completely different. The performance of the bcntimer is mainly determined by the PCI bus and the interrupt system in the kernel, since the remainder of the timer interrupt is embedded in hardware in the Atheros network card and its delay is insignificant to the delays experienced by the software. The interference of the parallel timers does not reside in the timer software construction and scheduling, however in the interrupt handling of the timers. This gives an indication that the stress factor lies with the interrupt processing and not within the timer structure. Since only the influence of parallel timers is verified, further performance degradation could result from a heavily loaded PCI bus and interrupts.

7.3.3 The hrtimer performance and dynticks or network transfers

The following comparison constitutes of the hrtimer with and without network load. The SWBA timer is not included in the comparison, since it has already been shown that its portrayed precision is lower than the precision of the hrtimer. In order to make the comparison complete, the results of a test run with dynamic ticks is added to the comparison. Figures 7.5 and 7.6 show the results for a 1000 µs and 250 µs timer interval respectively.

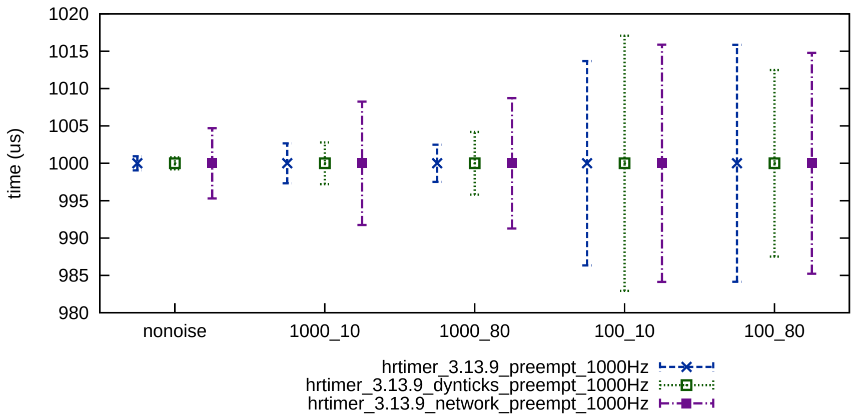


Figure 7.5: *3.13.9 hrtimer vs dynticks and network operation, 1000 µs interval*

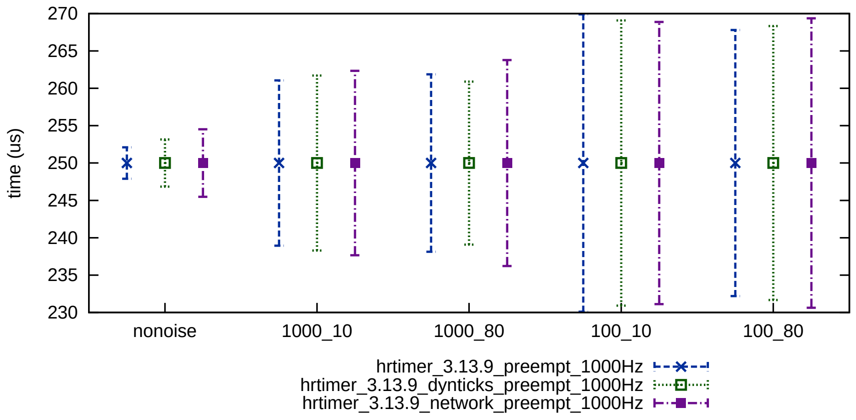


Figure 7.6: *3.13.9 hrtimer vs dynticks and network operation, 250 µs interval*

As expected, the figure clearly highlights the similarity between the hrtimer performance with periodic tick configuration and with dynamic tick configuration. As was mentioned before, the dynamic tick rate is not expected to influence the operation of the hrtimer and this figure shows that indeed it does not.

In order to verify the performance of the hrtimer in combination with a significant network load, which was said to be the main cause for timer latencies in [187] due to the softirqs, data was being sent as fast as possible over a wireless link during the timer analysis. The average transmission speed resulted to be between 45 Mbps and 98 Mbps. In both figures it can be noticed that there is indeed a negative influence from the network load compared to the results without network load, while there are no timers running in parallel. However, when activating the interfering timers, the effect from the network load is not noticeable anymore when comparing to the situation where no network transfer is present.

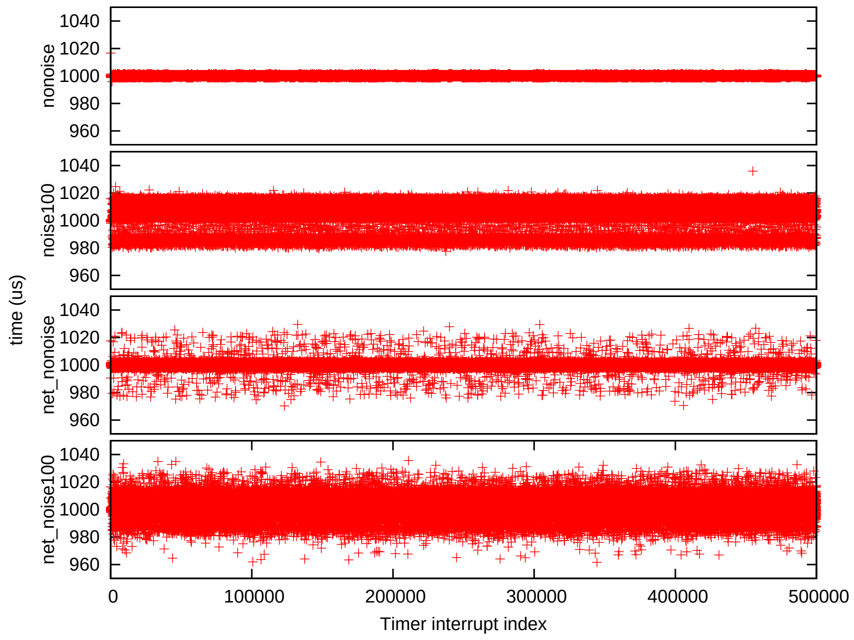


Figure 7.7: *inter-arrival time 3.13.9 hrtimer and network load, 1000 µs interval*

A different view on the inter-arrival time is shown in Figure 7.7, which depicts the inter-arrival times of the timers for each interrupt. The projected results are obtained from the performance analysis of hrtimer for kernel 3.13.9 without network load in the upper two figures for both nonoise and noise. The interfering timers consist of ten timers running in parallel with a start interval time of 100 µs. The bottom two figures depict the hrtimer with network load for both no noise and noise with timers at intervals of 100 µs. From the figures where there are timers in parallel, the distinction is not so clear. However, when comparing the figures where no timers are running in parallel, there is a clear difference between the case where there is no network load and the case where there is heavy network load.

Although a degradation of the timer performance is noticeable when increasing the system load by flooding the wireless interface, it does not automatically signify that the problem is caused by the employed softirq routine as claimed in [187]. In fact, this is rather questionable. The hrtimer softirq is one of the lowest priority softirqs and is therefore executed after all other softirqs, such as the net receive softirq, have completed. However, the hrtimer softirq is only used as an optimization, which checks whether timers are available to be triggered upon scheduling a new timer. Moreover, if the softirq is to blame the loss of precision, then the sum of all tasklets that are scheduled would need to trigger the same loss, since the softirq it employs also has a higher priority than the hrtimer softirq. The analysis of the actual cause of the performance drop is out of scope of this work, however, when receiving this much data, the wireless network card is generating quite a lot of interrupts over the PCI bus, around 8000 per second when no aggregation is used. This is equivalent to a timer running at an interval of 125 µs. Therefore, it is not necessarily the cause of softirqs that a degradation can be observed. However, it is clear that heavy network operation does influence the timer performance, if not in such a high order as timers that are running in parallel.

7.3.4 Proprietary real-time timer analysis

The following paragraph is focused in verifying the hypotheses that the scheduled timers can interfere with each other when having similar deadlines. In order to verify this, a proprietary timer structure is designed and implemented. Since more unused hardware timers are available on the IGEPv2 ARM platform, compared to the Alix platform, the OMAP2 architecture specific code was modified to support an extra clockevent, which can be utilized by the customized hptimer functionality. The hptimer functionality incorporates several mechanisms such that timers, scheduled on the same timer structure or even hrtimers, do not interfere with each other. First, in order to prevent concurrent use of the hardware timer between the hrtimer and hptimer, the hptimer is assigned a dedicated hardware timer per CPU. Second, the hptimer includes an acceptance check before scheduling new timers. The periodic timer is required to diverge from the already scheduled and established timers. This acceptance verification is based both on the timers periodicity and start time. If the check fails, the timer is rejected for scheduling on the current CPU and an effort is made to schedule it on a different CPU. If none of the CPUs allow the timer, the request to schedule the timer is rejected. Since the hptimer functionality is intended for a small number of timers that require a high precision, this restriction is not expected to cause issues.

The Figures 7.8 and 7.9 depict the comparison between the regular hrtimer and the custom made high priority hptimer for kernel version 3.2. The ARM platform provides two types of hardware timers, the gp timer and the more stable, but with a lower resolution, 32k timer. Measurements have been performed with both hardware timers. The hrtimer and hptimer results with prefix *ARM* indicate the usage of the

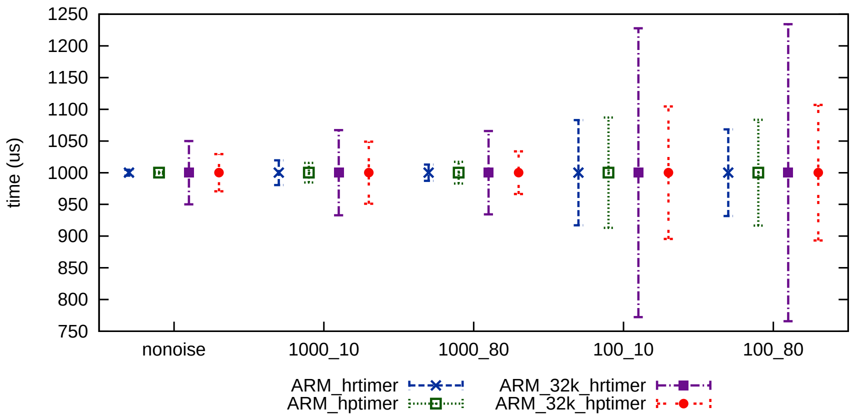


Figure 7.8: *Hrtimer and hptimer on ARM-Cortex A8, 1000 µs interval*

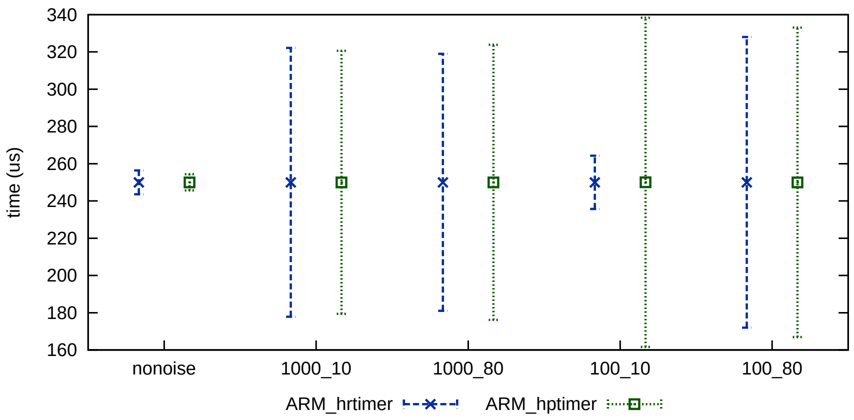


Figure 7.9: *Hrtimer and hptimer on ARM-Cortex A8, 250 µs interval*

higher resolution gp timer, while the results with as prefix *ARM_32k* employed the 32k hardware timer. For the timer tests with a 250 µs interval, the results for the 32k timer are not depicted, since it is not sufficiently precise to give any meaningful information regarding precision.

From the figures it can be seen that the performance of the hrtimer and hptimer are very similar. There are some variations, the one time the hrtimer shows a better performance, the other time is the hptimer better, however, this is due to random events upon which no control can be excised. When using the 32k hardware timer, it seems that the hptimer is performing slightly better than an hrtimer with the same clock source. However, the performance improvement is marginal compared to the performance of the hrtimer and not worth the effort of implementing another type of timer functionality in the kernel. These results indicate that in a regular system with a single CPU, there are no grounds for the hypotheses that the multitude of scheduled timers can deteriorate the timer performance.

7.3.5 Real-time Operating System timers

From the former results can be concluded that the hrtimer is regular and does not experience an extremely large jitter, even when other timers are stressing the system. However, it could happen that there is an outlier, which results in the missing of one or several timer triggers. A real-time operating system would provide strict deadline boundaries, thereby eliminating the outliers. The following measurement results depict the performance of the hrtimer in a Linux kernel patched with the RT-patch, the performance of the real-time core timer in a Linux kernel patched with the Xenomai real-time extension and the performance of a timer within an eCos RTOS.

The RT-patch [200], also called PREEMPT_RT, is a Linux project that focuses on providing patches for the kernel in order to enhance the real-time behavior. The enhanced kernel does not provide a hard real-time Linux system, however, aims to improve the performance of certain tasks that have a higher real-time priority. One of the changes that the patch employs is to convert most interrupt handlers into preemptible kernel threads, that is, the interrupt handlers have become processes that can be scheduled. A minimal interrupt handler processes the incoming interrupts and triggers the corresponding kernel interrupt thread. Thanks to the conversion, some locking mechanism that did require the disabling of interrupts can now use regular locking mechanism. In such manner, the time spent in interrupt context is shorter, allowing a faster IRQ handling. Since the hrtimer interrupt handling is not converted to a kernel thread and thanks to the reduction of the number of times interrupts are disabled, the hrtimer could exhibit an improved performance.

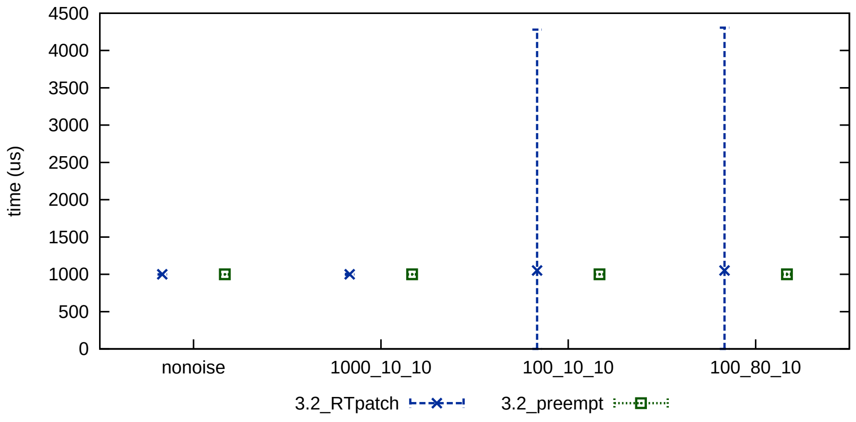


Figure 7.10: *Hrtimer interrupt interval time, RT-patch, 1000 μs interval*

However, note that the term real-time Linux is often used in the context of improving the responsiveness of user-space applications. Not only placing the interrupt handlers in kernel threads, but also the accompanied manipulation of spin_locks, the addition of priority inheritance and other real-time mechanisms ensures a completely different behavior of the system. This is also noticeable in the results (Fig.

7.10), where it is clear that the system modifications result in extreme jitter experienced when high rate timers are running in parallel. No performance measurements were executed with an interval of 250 μs, since the performance with 1000 μs already indicated a significant degradation compared to the original kernel.

Whereas the RT-patch is focused on improving the real-time behavior of the Linux kernel, the Xenomai real-time patch introduces a micro real-time kernel next to the Linux kernel. Thanks to the Adeos pipeline, the real-time core is the first to accept interrupts and can determine which interrupts the Linux kernel is allowed to receive. Instead of measuring the performance of the Linux kernel hrtimer, the timer embedded within the real-time core is monitored. The Xenomai patch was applied to Linux kernel version 3.10.32, which was the most recent patch available at the time of executing the performance measurements. The real-time timer is compared to the hrtimer of kernel version 3.13.9.

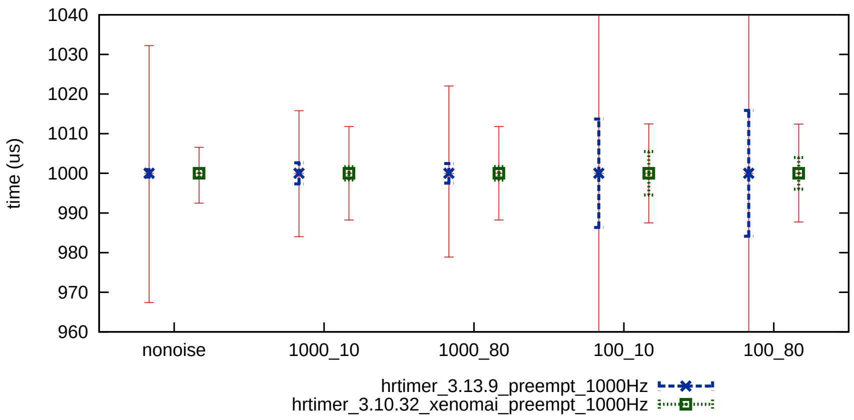


Figure 7.11: *Xenomai timer interrupt interval time, 1000 μs interval*

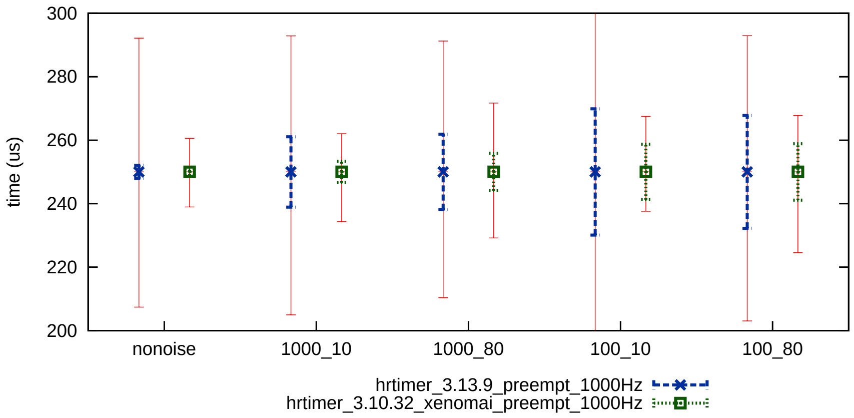


Figure 7.12: *Xenomai timer interrupt interval time, 250 μs interval*

Figures 7.11 and 7.12 depict the measured performance of the specified timers with a timer interval of 1000 µs and 250 µs respectively. The contents of the figures deviate from previous figures, such that the most relevant results are depicted. While the average timer interval and the 95% confidence interval are depicted in a similar manner as in the previous figures, the maximum deviation is added to the figure, as a straight red line, which forms the outer boundaries of the bar. The results concerning the average value and 95% confidence interval show a slight improvement in the jitter of the timer interrupt intervals. However, the major difference is the elimination of outliers when measuring the Xenomai real-time core timer performance. It can be noticed that the maximum interrupt interval of the hrtimer, running at a requested timer interval of 1000 µs, while having interfering timers running with a timer interval of 100 µs, exceeds the other maximum values. The exact values are not shown here for the clarity of the figure, but the maximum interrupt interval is 4716 µs and 4363 µs for interfering timers with priority 10 and priority 80 respectively. Likewise, does Figure 7.12 show that the maximum inter-arrival time of the hrtimer exceeds the posed boundaries when interfering timers are scheduled concurrently at a timer interval of 100 µs. The excessive interrupt interval has a value of 4460 µs. None of the timer measurements of the Xenomai real-time core exhibited such excessive behavior, even when performing a test for a longer duration.

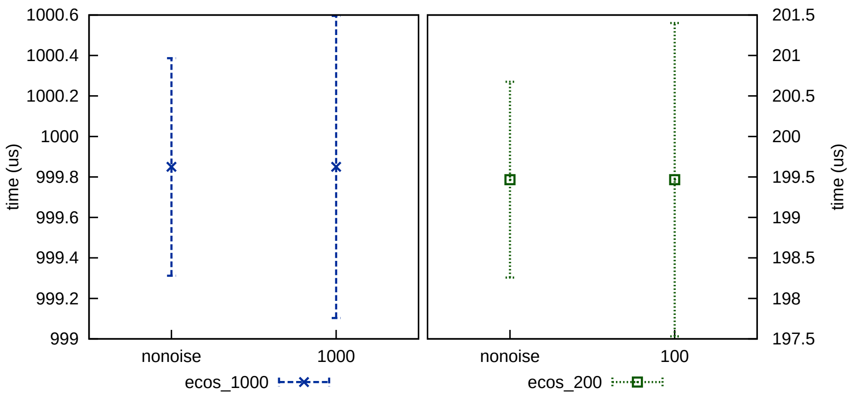


Figure 7.13: *Ecos interrupt interval time, 1000 µs interval*

All previous results were based, in one way or another, on the Linux kernel. The following measurement results give an indication of the timer performance within the eCos RTOS. The system is focused on running a single real-time application, possibly consisting of multiple threads, instead of providing multi-user support such as in Linux. Figure 7.13 depicts the results of a timer test that has been run on eCos, both for a 1000 µs and 200 µs timer interval. Since there are no real-time priorities associated to the interfering timers, only the timer interval of interfering timers is indicated at the x-axis. Note that the timer system of eCos is tightly bound to the operating frequency of the remainder of the system. Therefore, the lowest possible timer interval with the default system configuration is 10000 µs. In a first phase, the global system clock was decimated, such that timer intervals of 1000 µs were possible, hence the relative high timer interval value of the interfering timers. In order to retrieve measurement data of higher rate timers, the global system clock was again decimated, such that a timer interval of 100 µs was possible. This also allowed to test with a lower interrupt interval for the interfering timers. From the results, it is clear that the performance of an actual RTOS can significantly outperform that of the hrtimer performance within the Linux kernel, even in the most stressed situations. The maximum deviation within the 95% confidence interval ranges from 400 ns to 2.5 µs. Note that, since eCos does not provide such extensive hardware support, a different hardware timer has been selected to provide the test results, which has a worse performance compared with the timer that was used with the Linux kernel performance evaluations. Although the difference between the results is significant, they need to be placed in the right perspective. The Linux system provides interactive multi-user, multi-threaded support, running usually around 100 processes, while eCos is a single application OS, where in the current test application only ten parallel threads were running. The goals of the operating systems are different, where Linux tries to support everything that is required and more, while eCos lacks a lot of that support and requires some extra work to enable some functionality.

7.3.6 Conclusion

To conclude this timer performance analysis, it is clear that the beacon timer of the Atheros card itself is less precise, compared to the hrtimer of the Linux kernel, but it is hard to say whether this is a result of the accuracy of the hardware or the variable delays that occur during the signaling of the interrupt over the PCI bus. Also the coarser granularity of the timer might have an influence. As regards the hrtimer of the Linux kernel, neither the preemptiveness, the HZ configuration, nor the dynamic tick configuration have a significant influence on the performance of the timers, as was expected. The influence of a heavy network load degrades the performance of the hrtimer, however, several high speed timers running in parallel cause a much larger degradation in the performance. Furthermore, it can be said that the Linux hrtimer can be deemed efficient, even although all system timers make use of the same timer structure. When compared with an isolated timer structure, specifically designed for high priority timers where measures are taken to prevent coincidental timer deadlines, the performance of the hrtimer is similar.

Despite the excellent average performance of the hrtimer, the timer happens to lack hard real-time constraints. As regards timer expirations, this translates into an average interval almost equal to the requested timer interval, however, occasionally the timer interval can exceed even twice the requested interval time. Attempting to resolve the issue by means of the rt-patch shows that the patch transforms the system drastically, resulting in timer performances that are not comparable to the original kernel. On the other hand, the Xenomai real-time Linux kernel extension does provide a reliable timer, which does not exhibit the excessive interrupt intervals that represent the outliers. A dedicated real-time OS, such as eCos, outperforms significantly the Linux kernel timers. However, unfortunately, hardware support is limited and Atheros wireless devices are not supported within eCos.

Note that these results are related to the system on which they were performed. An SMP machine will for example exhibit a better performance, since timers can be spread out over all the processors. However, in order to do a performance analysis of the timer itself and not of the combined system, a single processor system should be selected, as has been done in this section.

7.4 Reliable TX on IEEE Std 802.11n hardware

The previous sections discussed the stability and performance of the identified timer sources, which is a prerequisite for the remainder of this chapter. The required adaptations to the Linux kernel wireless subsystem are discussed in the remainder of this chapter, such that a reliable and precise slotted transmission arrangement with commodity hardware is made possible. Before describing the required changes in the Linux kernel to ensure a reliable transmission on standard hardware, this section provides some essential background on both the hardware and the method. First a description of the hardware, the relevant Linux kernel components and their interaction is provided. Afterwards a number of common misconceptions are discussed. Such misconceptions, such as the sole consideration of WMM in order to provide a reliably timed transmission, result in only partial or flawed designs. The last two sections deliberate on the methodology to ensure such precisely controlled transmission schedule on COTS hardware.

7.4.1 System description

At the time of conceiving this implementation, Linux 2.6.38.2 is the most recent stable kernel version and AR9220 is one of the IEEE Std 802.11n compliant chipsets that are commercially available on wireless cards. The Atheros AR9220 chipset is a member of the AR9002 family. Only a couple members of the family are discussed, which are all related to the same wireless core, but provide different interfaces towards the Linux kernel. The Linux kernel driver for these cards is either ath9k for a Mini PCI or PCIe interfaces type, or ath9k_htc for a USB interface type. Both drivers make use of the Linux kernel SoftMAC principle, that is, they register the detected network card to the mac80211 module (See Section 6.4.4). Since the considered use case consists of a network conferencing in which a central server is to distribute content to the clients, the approach focuses on an Infrastructure BSS (AP mode) type of network. Note that the traditional Linux kernel wireless subsystem does not provide support for AP MLME handlers. A user-space application, such as hostapd, is responsible for processing and responding to the corresponding management messages. The remainder of this section will elucidate the hardware specification and operation of the system in so far as this has not been done already

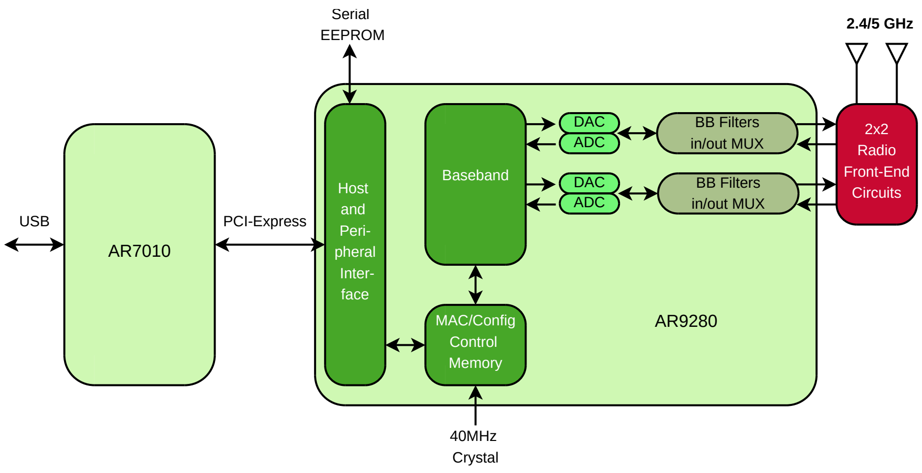


Figure 7.14: *Atheros AR9002U (AR7010 + AR9280) block diagram. (Ref: Adapted from Atheros AR9002U (AR7010 + AR9280) Product Brief, 2007 [221].)*

in Section 6.4.4.

Atheros chipsets such as AR9220, AR9280 and AR9002U are all part of the same product family. The core wireless functionality of all these devices is precisely the same. The difference between the chipsets is situated in the interface towards the controlling system, such as a Linux operated PC. Both the AR9220 and AR9280 provide a user interface over the PCI bus, although the form factor varies; the AR9220 offers a Mini PCI bus, while the AR9280 offers a PCIe interface [220]. The AR9002U provides a USB interface to control the chipset [221]. Interesting to note is that the AR9002U is internally constructed out of the AR7010, which is mostly a general purpose CPU, and the AR9280. The AR7010 communicates with the AR9280 over the internal PCIe bus and provides a translation of the commands that are received over the USB bus into read and write operations over the PCIe bus. The block diagram of the internal structure of both the AR9280 and AR9002U is depicted in Figure 7.14.

The AR9220 and AR9280 provide access through the PCI bus to a section of memory where parameters are stored that control and provide feedback on the MAC operation. Even the transmission and reception of packets is performed by setting the configuration registers. A pointer to the memory address of the descriptor of the first packet in a queue is written in the configuration. After the chipset has received the signal to start the transmission, it fetches or writes that data from/to the provided memory address by means of a DMA transfer. Due to the different approach of communication with the devices, the AR9220 and AR9280 chipsets are controlled by means of the ath9k Linux kernel driver, while the AR9002U is controlled by the ath9k_htc Linux kernel driver. Whereas the ath9k driver has direct access to all configuration registers and provides the possibility of using the chipset in a non-standardized manner, the ath9k_htc driver mostly provides a command interface

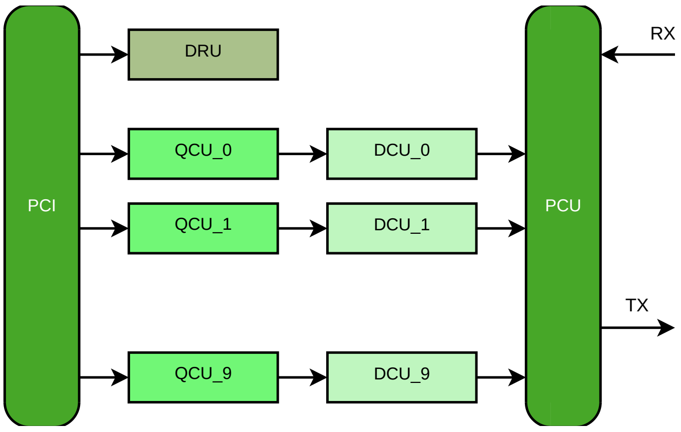


Figure 7.15: *Queue structure of AR9220*

Figure 7.16: *Queue structure of AR9220. (Ref: Adapted from Atheros AR9280 Single-Chip 2x2 MIMO MAC/BB/Radio datasheet, 2009 [222], Figure 3.1, page 14.)*

to the USB commands that the integrated AR7010 is able to understand. The customization is therefore limited when working with the AR9002U. The ath9k_htc is required to load firmware into the AR9002U, that is, binary code that the AR7010 is able to understand. At the time of conceiving the idea of customizing the transmission operation of wireless devices, this firmware was closed source proprietary code, which performed similar actions as the open source ath9k driver did in the Linux kernel. Therefore, it was not possible to conduct experiments with this chipset and was therefore excluded from this work. However, recent developments persuaded Qualcomm/Atheros to make the firmware code open source, allowing the customization of the AR9002U operation. This creates opportunities for future research, since the firmware running on the internal CPU of the chipset is not interfered by processes or interrupts that are not considered relevant to the network operation.

In the remainder of this work, only the AR9220 will be considered, since the employed development platform provided a Mini PCI interface and not a PCIe interface. The internal operation of the AR9220 chipset, more specifically, the transmission and reception architecture is depicted in Figure 7.16. As specified earlier, both the transmission and reception mechanism need a pointer towards a section of reserved memory in the DMA memory range of the system. Both the DMA Receive Unit (DRU) and Queue Control Unit (QCU) components make use of this pointer to transfer to and from the host system by means of DMA. The DRU (DMA Receive Unit) is the module responsible for the transfer and signaling of received packets. The unit triggers a PCI interrupt upon completion of the reception of the packet. The ath9k driver should handle the interrupt and initialize a new pointer to memory in the DMA range.

The transmission queue system is somewhat more complex due to the provision of

QoS. There are 10 QCUs (Queue Control Unit) of which each is linked to its own specific single DCU (Distributed Coordination Function(DCF) Control Unit). The QCUs are assigned priorities, of which QCU 9 has the highest priority. Each QCU has a pointer to a TX descriptor, which is the first element of a linked list. The TX descriptor contains a pointer to the data (the actual packet that is being sent) and additional MAC control information. When a TX descriptor is written to a QCU and the conditions of the scheduling policy are fulfilled (the queue is triggered if necessary), the QCU transfers the frame to its associated DCU. The QCU provides several possible scheduling policies, that is, event-based, time-based or ASAP. In the case a trigger or timer is required, it needs to come from the chipset itself, external triggers are not considered valid sources.

The DCU (Distributed Coordination Function Control Unit) provides the DCF functionality (Distributed Coordination Function), such as backoff, waiting for an idle channel, etc. When a packet is ready to be transmitted, the DCU signals the DCU arbiter that a packet is waiting to be transmitted. The DCU arbiter decides which DCU is allowed to send its packet to the PCU (Protocol Control Unit), according amongst others their respective priority. After the actual transmission, the DCU writes the status in the TX descriptor, which is a pointer to the host memory. The PCU is responsible for buffering TX and RX frames, automatic acknowledgement transmission, RTS, CTS, carrier sensing, etc. An interesting feature the AR9220 chipset provides is the diagnostic register memory area. This configuration register allows to manipulate certain parts of the PCU functionality, such as disabling the transmission of acknowledgements, forcing a clear channel assessment, etc.

The driver that implements the control of the AR9220 chipset is ath9k in the Linux kernel. This driver controls the wireless card by writing to its registers which are accessible through the mini-PCI bus. In a most simplified manner, it could be said that the driver accepts packets, arranges them in a linked list, passes the address of the first element of that list to the appropriate QCU and signals the hardware that new packet information has been written. Data is sent according to its QoS type in QCU 0 to 4, which are configured to send as soon as data is available. Note that we are working with IEEE Std. 802.11n compliant hardware and the Linux kernel code follows the latest version of the standard, which is the reason that IEEE Std. 802.11e is already incorporated in both. In reality, there is a lot of configuration and preparation to be done by the ath9k driver, such as allocating memory, manipulation of several linked lists, combining packets that should be aggregated by the hardware, calculating the background noise, storing physical parameters that are measured during transmission and reception for diagnostic purposes, manipulation of encryption keys, etc. The ath9k driver also accepts configuration values from the upper layers, that is, the mac80211 module, where global configuration is performed through the ieee80211 API, while per packet parameters can be set in the skb callback and information section, which is part of the packet coming from the mac80211 module.

While one of the tasks of the ath9k is to configure the hardware, another major task is to capture any interrupts that are triggered by the hardware and handle them accordingly. Interrupts could signal the transmission or reception of a packet, accompanied with its error codes, but it could also signal other events. For example, in AP mode the hardware signals an interrupt at the expiry of the SWBA (SW Beacon Alert) timer, upon which the ath9k driver prepares the next beacon packet and places it into the beacon queue, which is QCU 9. This queue is configured such that when a packet is ready to be sent, it blocks all lower priority queues from entering new packets. Since Queue 9 has the highest priority, all queues are blocked until the beacon packet has been sent. The beacon queue is timer triggered and waits for the occurrence of the DBA timer (DMA Beacon Alert), which is an internal timer of the AR9220 chipset, upon which the DCU will try to send the beacon.

The remainder of the Linux kernel wireless subsystem has already been discussed in Section 6.4.4. Note that since the mac80211 module does not provide any management handlers for the Access Point, a user-space application is required, named hostapd. It provides methods to configure the network interface in Infrastructure BSS mode, provides management handlers for this mode, as well as encryption, association, authentication and timings. The program is used at the node that is acting as AP and configures the respective layers by means of netlink and ioctl calls. Note that, at the time of implementing the slotted transmission mechanism, the transformation from ioctl to netlink calls had not yet been completed. Hostapd also provides other types of interfaces for operation in various environments, however, this is considered to be out of the scope of this work.

The management packets are received through a monitor interface, which is setup during initialization. The transmission of management response packets is also performed through the same monitor interface. Hostapd keeps a strict control on the timing of incoming responses and discards any responses that are too late.

7.4.2 Common issues and misconceptions

It is common in the related works that either it is assumed possible to disable the CSMA/CA functionality of commodity hardware, the modus operandi is left to the imagination of the reader or they claim to be able to exert TDMA access mode by relying solely on the manipulation of QoS parameters. Unfortunately, the IEEE Std 802.11 commodity hardware is specifically designed for CSMA/CA operation and does not provide, until so far, a switch that could disable this mode and only manipulating the QoS parameters does not suffice. However, it is possible to reduce the influence of the standard behavior of the commodity hardware. Some of the actions that are required to reduce the influence of the CSMA/CA mechanism is listed in [175]:

1. Eliminate automatic ACK and retransmission

2. Eliminate RTS/CTS exchange
3. Eliminate virtual carrier sense (NAV)
4. Control PHY Clear Channel Assessment (CCA)
5. Control transmission backoff

The previously mentioned related work resolves the first two action points by using the hardware in monitor mode, while the other actions are handled by register manipulation. Because the paper refers to IEEE Std 802.11b/g hardware, or because the authors work with monitor mode, there was no need yet to enumerate certain other actions that need to be taken when discussing IEEE Std 802.11n hardware or when working in AP mode. Some of these actions are the disabling of Block Acks, disabling of rate control, manipulating the beacon queue properties, etc. However, a most important issue is the need to disable receptions when no receptions are expected. This is an issue that none of the related work discusses, except in [184], where it was noted that the protocol exhibits a decreased performance in an environment with other WiFi devices, although it is performing as expected in an isolated environment. The issues experienced in an environment with other active WiFi devices does not come from the availability of foreign data transmissions, the problem is rather caused by the multiple probe requests that are broadcast. Probe requests are low rate transmissions and therefore have a longer duration. When the AR9220 is receiving such a message, all scheduled transmissions have to wait upon the reception of the packet.

7.4.3 Hardware controlled operation

One of the options that the Atheros AR9220 hardware provides is the usage of the QCU's scheduling policies, where by means of the time triggered scheduling policy packets can be sent out at a regular interval through the use of the DMA beacon alert (DBA) timer. The DBA timer is a timer with a precision of 128 µs and therefore has a sufficiently precise resolution for the specified use case. The timer triggers the queue system such that available packets are marked ready to transmit and the transmission continues until the end of the queue is detected. In combination with a ReadyTime limit, slots can be defined in which transmissions are allowed. Such method is employed in [223], where the detail of control is even extended towards the application layer such that by opening an interface in TDMA mode, all packets sent through that interface are using this configuration. Although the timer provides a means to slotted operation, there is no control regarding the slot allocations, neither is it possible to know where the slot boundaries are located. The method allows to send packets in a CSMA manner during specific time slots, which is fine for some applications. However, for some protocols, such control is too limited and therefore require an additional control mechanism that enqueues packets at regular intervals if management and data packets are to be sent in specific slots.

This would require some extra timer source and thus redundant functionality. Note that no performance analysis was performed on this timer in Section 7.3, since this timer is located internal in the Atheros hardware, without any interface towards the host system. As such, the methodology used to compare the other timers is not be applicable to this timer.

7.4.4 Software controlled operation

Depending on the operational mode that the network device is working in, different modifications are in order to implement a controlled slotted transmission scheme. Since the targeted environment is infrastructure based, it is more convenient to modify the AP mode than any other mode, since it already provides a lot of functionality that is required, but which in some cases needs to be modified.

The goal is to control the transmissions as much as possible with an IEEE Std 802.11n compliant device. Since the DCF functionality is embedded in the wireless network device, it is required to provide a reliable time triggered packet transmission towards the wireless network device by buffering packets coming from the upper layers and forwarding them one at a time to the hardware when triggered by a timer source, such as the works in [172] and [183] indicate. At the time of conceiving this implementation, Linux 2.6.38.2 is the most recent stable kernel, of which a superficial timer analysis indicated the suitability of the hrtimer for the targeted use case. The analysis consisted of a comparison of the hrtimer and the SWBA timer on a regular PC with an SMP architecture for all possible configurations regarding preemptiveness or HZ value, however a detailed comparison with a custom timer functionality, on different (single CPU) platforms, with different improvements to achieve real-time etc. was done only for kernels 3.2 and 3.13.9, which is discussed in Section 7.2. Based upon those first impressions, the hrtimer was selected as timer source.

Due to the integrated functionality of the network device, the jitter exhibited by the timer source will be augmented by the default network functionality of the device when transmitting. To elevate the effects of the functionality of the chipset, some of this functionality needs to be bridged by disabling certain features such as the automatic acknowledgement transmission, the carrier sensing, RTS/CTS handshaking, minimizing backoff before transmissions, etc. The disabling of power saving and Block Acks also enhances the precision of the packet transmission, by eliminating variable factors. Automatic rate control is also of course out of the question when predictability is required.

The keyword of the system is reliability. Therefore, management packets also should need to be sent in a controlled way. Most management handlers for AP mode are located in the hostapd user program. The management handlers provide the functionality for the association procedure. For the targeted use case, which is to validate the manageability of a slotted transmission over commodity hardware, there are no stringent requirements on the association procedure, since it can be considered as the initialization phase of the system. A system beyond this use case needs to make sure that the stations first listen for the beacons sent by the AP, instead of immediately starting to send probe requests, since the stations should follow the slot allocations of the system. Since this work focuses on the development of a new type of protocol, where such an extensive information exchange during the association is not required, the association procedure itself as well as the contents of the management packets can be simplified.

7.5 Implementation of controlled transmissions

It has already been mentioned before that a controlled slotted transmission is not supported in the ath9k driver. Moreover, the Atheros AR9220 wireless network card only supports IEEE Std 802.11n compliant medium contention access. Therefore, it is required to optimize the network card for a low latency transmission and perform most of the packet transmission triggering in software. Since most data is originating from the AP in this case study, the current validation procedure allows the AP to transmit in a slotted manner, while the STA simply listens. In a further stage, this needs to be extended to a full TDMA algorithm, where the stations also send their own data back. This implies that most changes need to be done at AP side and just a few at the station side.

Due to the stringent timing requirements that hostapd enforces, association with the modified slotted transmission ath9k driver is hard or even impossible in the 5 GHz band under certain circumstances. Due to the slotted transmission, system delays and the manner in which hostapd handles notifications, the association procedure is able to complete successfully, however, due to a timeout notification a disassociation message is sent nevertheless by hostapd even after the association has been completed. Therefore, the first step into the direction of a custom association procedure is required. Moreover, due to the fact that hostapd is a user-space application, it needs to use a variety of interfaces towards the kernel in order to provide the required functionality. All these facts combined led to the decision to create a simplified kernel module that provides sufficient functionality such that it can replace hostapd. The next subsection clarifies the migration of the hostapd user program towards the AP kernel module, while the following subsection goes into detail regarding the modifications to the ath9k driver. The final subsection discusses the obtained results from the implementation.

7.5.1 hostapd

Since the Linux kernel mac80211 module does not provide MLME message handling for AP mode, a user-space application such a hostapd is required to enable nodes to operate in AP mode. Since the application resides in user-space, while requiring

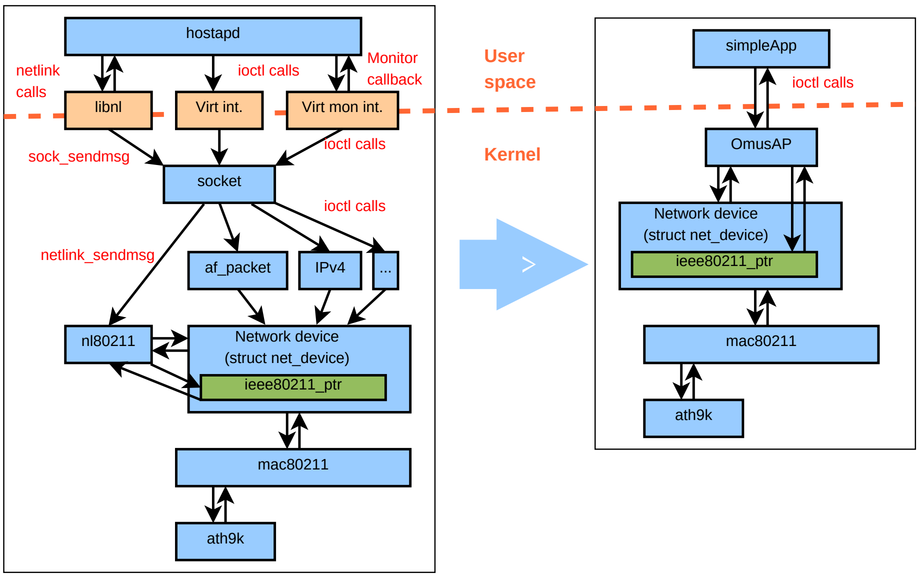


Figure 7.17: *Migration from hostapd application to AP kernel module*

packet arrival information from the mac80211, a rather complex interface towards the kernel is provided. The standard messaging and interface structure of the kernel that hostapd needs is depicted on the left side of Figure 7.17. Note that this block diagram is a simplified diagram, not showing irrelevant components, such as the L2 packet socket interface, through which monitor packets are transferred to user-space. The diagram is based on the composition of the hostapd application during the development of the slotted transmission algorithm. While the diagram shows the requirement of using ioctl calls and a monitor interface, both are not required anymore with the newer versions, provided that the Linux kernel wireless subsystem assures the required functionalities. Instead of listening to the monitor interface for incoming messages, the hostapd application registers itself as listener for a certain class of netlink family events. The monitor interface can still be used as a backup solution if not sufficient functionality is provided.

The simplified kernel module, the AP kernel module, depicted on the right side of Figure 7.17, performs the basic functionality that such module should provide. Such actions include the initialization of the NIC into AP mode, based upon parameters passed on from user-space to allow for easy configuration, and the handling of incoming management requests. The module provides a filesystem through which ioctl calls can be made from a user application. Through the supported ioctl calls, the AP can be started with the passed parameters and stopped. Note that the AP module interacts directly with the network device and the cfg80211_registered_device structure , thereby limiting the function call stack. Neither encryption nor authentication are supported within the AP kernel module in order to provide solely the most basic functionality.

The initialization phase includes the configuration of the network device, the deauthentication of previously authenticated nodes, the verification whether the parameters that were provided from user space match with the capabilities of the hardware, and the construction of a beacon message containing all capabilities and configuration settings of the AP and writing this beacon message to the hardware. The configuration of the device entails AP mode operation, frequency settings, basic rates, SGI, Short Preamble, RTS threshold, fragmentation threshold, QoS parameters per queue, such as TXOP, CWmin, CWmax, AIFS, etc. An additional functionality that has been added to the ath9k driver while implementing the controlled transmission, is a MAC filer. The MAC filter checks upon arrival of a packet whether the MAC address is one of the expected MAC addresses, otherwise the packet is dropped. The addresses that are supported is also configured during the initialization phase of the AP kernel module.

The handling of management packets is supported by placing hooks in the mac80211 module, not unlike the monitor mode hooks. Since the AP kernel module is constructed to be a loadable module, a global method is provided where the function reference that the hook calls can be set and reset. Upon module initialization, it registers its own message handler function as being the function the hook calls. Through the use of those hooks, the AP module receives the management packets that were either not handled or passed on to the monitor interface anyway by the mac80211 module, as well as transmission acknowledgements. The packet handlers that are supported within the AP kernel module are probe, association and authentication related messages, and action and beacon packets. The management packets currently have the same format and functionality as the standard IEEE Std 802.11n management packets in order to be able to test with standard stations. In a later stage, the management format and the number of management packets can be modified to support a more simplistic association procedure.

7.5.2 ath9k

Whereas the previous section discussed changes to the association functionality, the core modifications that allow reliable and deterministic transmissions are discussed here. The ath9k adaptations comprise of the resolving of a number of issues. Such solutions include the methodology and parameter transmission queue adjustment, association method and reception period adjustment, and the manipulation of several mechanisms, such as Block Acks, unicast acknowledgements, RTS/CTS transmissions, Carrier Sensing, rate control, etc.

As already depicted in Section 7.4.1, the AR9220 provides several queues to support Enhanced Distributed Channel Access (EDCA). The ath9k driver provides support for the enqueuing of a packet based on its QoS class. A packet that is received from the upper layers is first processed for its meta-data and a transmission descriptor is created for the packet. Based on the Access Class of the packet, the packet is enqueued in the appropriate hardware queue and the hardware is notified of the presence of new data.

The modified implementation, which allows a controlled transmission scheme, allows the preprocessing of the packet, the construction of the transmission descriptor, etc. to be performed after which the packet is temporarily stored in a buffer instead of enqueuing it in a hardware queue. The packets are placed in the hardware queue according to a fixed rate, that is, the action is performed as a response to a timer trigger. The trigger is a result of the scheduling of a hrtimer with a timer interval of 256 µs. When packets of 1500 bytes are being sent by the application layer, the theoretically available throughput would reach ∼46.8 Mbps.

Note that the adjustments ignore the specified Access Class (AC) of the packet. All packets are placed in a single hardware queue. As a result the modified system does not experience transmission queue switching delays, prevents internal collisions between different ACs, and the packet transmission is made more deterministic. All transmissions exhibit the same transmission properties as a result of the same queue properties, such as CWmin, CWmac, AIFS, TXOP, etc. Such parameters, as well as the channel configuration and HT parameters, are passed on by the AP kernel module during configuration time. All control regarding QoS can be performed in software by first allocating slots for higher priority transmissions. However, QoS control is implementation dependent of the TDMA algorithm, which is considered future work.

Note that the beacon transmission mechanism in the original ath9k driver results in the blocking of data transmissions during around 3 ms before the beacon transmission. The SWBA timer interrupt triggers the final preparation of the beacon packet and the placement of the beacon packet in the beacon queue. Beacons are sent in the highest priority queue, which is configured to block all lower priority queues. The SWBA timer is configured to fire around 3 ms before the internal DMA beacon alert (DBA) timer is configured to fire, hence the 3 ms blocking of all data transmissions. In order to prevent the blocking action when sending beacons, The scheduling of beacons is integrated in the packet scheduling. Beacons are scheduled to be transmitted in the same queue as data and a slot is allocated every 400 timer interrupts to send a beacon packet instead of a data packet, resulting in a beacon interval time of 102.4 ms.

As already mentioned in the previous section, the association procedure is maintained, since the focus of this work is on the controlled transmission. Keeping a similar association procedure allows to employ almost IEEE Std 802.11n compliant stations. The difference with the original Linux system is the replacement of hostapd by the AP kernel module. However, the ath9k driver is ignorant of the actions that are performed on higher layers. This implies that all management packets are considered as regular data packets and thus sent out according to the timer interrupts. The management packets are therefore interleaved with the data and occupy slots that would otherwise have been used for data packets. Since the current setup uses a single AP and a single station, all management packets are sent during the association procedure, which is the first phase, after which data can be sent during the second phase. In a later stage, this should be modified and management packets should be allocated in specific management slots and thus not interfere with the data transmissions.

Thanks to the different phases, the reception of packets can be shut down after the association phase. Since the medium is free to be used by any wireless node, following the medium access rules, a lot of traffic can be expected. Each reception prevents the hardware from transmitting, thereby causing considerable indeterminism in the transmission pattern. Since this work focuses on the reliable and deterministic transmission of packets, the reception of packets is disabled after the association has completed. In future work, this can be done in a more flexible manner and either a series of slots need to be considered as a transmission period where reception is not allowed, or the reception needs to be turned off (and cut off in case a reception is ongoing) each time a transmission needs to be scheduled. The work in [167] makes use of an interesting method, which can prove to be useful for this use case.

In order to enhance the timing control, certain features of the IEEE Std 802.11n protocol are disabled, such as Block Acks, power saving capability, aggregation, etc. The rate adaptation protocol of the mac80211 is completely ignored and the rate information is set to a fixed value of MCS 5 while creating the transmission descriptor. A variable rate would induce indeterminism, which is counteracting the targeted result. Note, even beacon packets are transmitted at MCS 5, otherwise it would take longer than 256 µs to transmit a beacon at the basic rate. Of the range of possible (re)transmission rates, only a single rate is specified, preventing the automatic retransmission with a lower rate. Even more, retransmissions of the initial rate are disabled by manipulating the retransmit count. The RTS/CTS bit is set such that no RTS/CTS is required, indifferent from the RTS threshold, and QoS data packets are sent with the No Ack bit set.

Although each of the previously discussed modifications provide an essential modification, the sum of all modifications is still insufficient to guarantee a deterministic transmission scheme. The internal registers of the AR9220 allow modifications on a more basic level of the MAC operation. The PCU diagnostic register allows to specify non standardized operation, such as prevention of automatic ACK transmissions and the disabling of virtual carrier sensing. Since the duration field of each packet, not destined to the current node updates the Network Allocation Vector (NAV), which determines whether or not a transmission is allowed, the disabling of the virtual carrier sensing is crucial. Another crucial parameter allows the device to assume the medium is clear, that is, Clear Channel Assessment (CCA) is not supported when this parameter is set. In order to ensure a timely transmission, the usage of this parameter is necessary. An implication to using this parameter is the loss of packets due to collisions on a busy medium, since carrier sensing is not being considered.

Since most changes occurred at the AP side, there are but a few modifications done at the STA side. Some of these modifications are vital, since the capability of a node to provide some features could influence the transmission pattern at the AP side. The functionality that needs to be disabled is amongst others, power saving capability, Block Acks and aggregation.

Since the reception procedure of the ath9k driver sets the timestamp to the system time at which the packet was processed in the driver, tcpdump is not able to see the actual time the packet was received. Therefore the actual mac timestamp at the reception is placed in the packet data in order to be able to analyze the packet arrival times, however, this is just for validation purposes and has no effect on the operation itself.

7.5.3 Reliable TX Performance Results

In order to make a performance analysis of the modified Linux kernel, a pair of nodes is used where one is configured as AP, while the other node is in station mode. Both nodes comprise an AMD Sempron(TM) 2400+ CPU, the necessary memory and a Mini PCI wireless Network Interface Card (NIC), containing an Atheros AR9220 chipset. As has been mentioned in the previous section, the operation of the wireless communication system is comprised of two phases. The first phase is the association phase where the AP kernel module is loaded and the application to start the AP mode is executed at the AP side, whereas the node that operates in station mode performs a regular association to the available AP. The second phase constitutes to the transfer of data. After the association phase a tcpdump application is started at the station side and a proprietary application, which allows the transmission of data at the fastest possible rate towards a MAC address, is started at the AP side. The packet is sent directly to the MAC layer. As such, the entire IP layer is bypassed in this manner and therefore also the ARP mechanism, which only adds to the complexity of the system, making it harder to determine the performance of the section that is under test. The received data is captured at the receiver side and stored for processing, where the inter-arrival times are determined by analyzing the receive times embedded within the received data packet.

Two different environments were defined to test the modified Linux kernel, the 2.4 GHz band and the 5 GHz band. The behavior in the two bands is so different due to the multitude of networks operating at the 2.4 GHz band at the test location, while at the most only a few interferers were present in the 5 GHz band. While most performance evaluations were based on longer duration measurements, also some short measurement intervals were defined. All tests have been performed with HT+, MCS 5 and the application transmits packets of 1500 bytes each.

The performance evaluations showed similar results in identical situations. Therefore only the three most interesting cases have been depicted in this section. The first two test are preformed in the 5 GHz band (channel 36) and the duration of the two tests was one hour and one minute respectively. The evaluation of the first test resulted in a total of 14027005 received packets and 517 missed packets, which amounts to a loss of 0.003%. A missed packet is defined as a packet that has been

Deviation range (µs)	Number of deviations
$[1, 2[$	0
$[2, 4[$	1486019
$[4, 6[$	23279
$[6, 8[$	2408
$[8, 10[$	1046
$[10, 20[$	4901
$[20, 30[$	7010
$[30, 40[$	502
$[40, 50[$	40
$[50, 100[$	113
$[100, 150[$	0
$[150, 200[$	0
$[200, 300[$	1

Table 7.1: *Deviations larger than 1 µs for the first test*

transmitted, but has not been received by the receiver. The average arrival time between two packets is measured to be 255.995 µs. Note the precise reception at the station side, since the interrupt interval of the hrtimer at the AP side is configured to 256 µs. The minimum time between two packets is 194 µs and the maximum time between two packets is 535 µs. The standard deviation of the time between two packet arrivals is equal to 1.19356 µs and therefore 99.7% of the samples fall within the mean value +/- 3.6 µs. Of those that have a deviation higher than 4 µs, most of them exhibit a value lower than 30 µs. All deviations larger than a µs can be found in text form in Table 7.1. The first column indicates intervals of the deviation from the average inter-arrival time in µs, while the second column depicts the number of measurement values that fall within the category. In total there are 12186 packets that exhibit a deviation which differs more than 10 µs compared to the average arrival time, which amounts to 0.087% of the total number of received packets.

The results of the second performance evaluation are measured with the same parameters as the previous test, except, instead of running the test for an hour, the measurement lasted only for a minute. Such measurement also provides information whether the performance of the modified kernel is equal, even over short periods of time. The measurement results indicate in total 233938 received packets and only 17 missed packets, amounting to a packet loss of 0.007%. Like in the previous test results, the average inter-arrival time of the packets is equal to 255.995 µs. Therefore, even during short periods of time, the hrtimer exhibits an excellent performance, showing no signs of startup discrepancies. The minimum measured arrival time is 203 µs and the maximum is 307 µs. The standard deviation of the time between two packet arrivals is equal to 1.28414 µs and therefore 99.7% of the samples fall within the mean value +/- 3.85 µs. Table 7.2 depicts a textual representation of deviation intervals towards the average inter-arrival time and the number of mea-

Deviation range (µs)	Number of deviations
$[1, 2[$	0
$[2, 4[$	39847
$[4, 6[$	1427
$[6, 8[$	22
$[8, 10[$	9
$[10, 20[$	134
$[20, 30[$	23
$[30, 40[$	2
$[40, 50[$	0
$[50, 100[$	2

Table 7.2: *Deviations larger than 1 µs for the second test*

surements that are contained within the respective deviation intervals. In total there are 137 packets that exhibit a deviation which differs more than 10 µs compared to the average arrival time, which amounts to 0.059% of the total number of received packets.

Although the final test result depicted in this section considers a test of only five seconds, the information contained within these five seconds is very useful. The test was performed in the 2.4 GHz band (channel 6), where the test system was subjected to a vast amount of external interference, that is, transmissions from other systems that are not controlled by the test system. Like in the previous two tests, HT+ mode, MCS 5 and a packet transmission of 1500 bytes was employed. In total 17019 received packets were measured and 2632 lost packets were detected. The total loss amounts to 15.5% of the received packets. The average inter-arrival time of the packets was measured to be 255.997 µs, whereas the minimum inter-arrival time is 228 µs and the maximum is 283 µs. The standard deviation was determined to be 1.16553 µs.

As can be noticed by the results, using the slotted transmission scheme in an environment with a lot of interference results in a vast amount of missed packets. However, the stability of the inter-arrival time of the packets is quite satisfactory, the standard deviation is around 1.2 µs, hence 99.7% of all inter-arrival times are within a deviation of 3.6 µs. Even of the remaining 0.3%, most of them are positioned within a deviation of less than 30 µs. Since the transmission duration of a data packet of 1500 bytes at MCS 5, HT+ is around 180 µs and the available slot time is 256 µs, this is still within acceptable boundaries.

7.6 Conclusion

This chapter focused on the transmission control on IEEE Std 802.11n compliant COTS hardware, that is, a non-standardized operation of the COTS hardware.

More specifically, the chapter was focused on the challenge to create a reliable and deterministic slotted transmission scheme in an Infrastructure BSS type of network. Two tasks were identified in order to achieve such feat. The first task was the analysis of the performance of the available timers in the Linux kernel and NIC hardware. The analysis was intended to determine which timer is sufficiently precise to guarantee a precise transmission. The second task was to modify the relevant parts of the Linux kernel wireless subsystem such that a deterministic transmission is made possible.

The results of the timer performance analysis section demonstrated the impressive deterministic behavior of the Linux kernel hrtimer. Even when multiple timers are running concurrently, thereby stressing the interrupt system, the timer shows an acceptable performance deviation. The fact that multiple system timers make use of the hrtimer structure does not deteriorate its performance, since a timer structure that is dedicated to only some high priority timers, and is scheduled on a different hardware timer, shows a similar performance as the hrtimer. The impact of a heavy network load results in a deterioration of the performance of the hrtimer, however its influence is limited in comparison to the timers running in parallel.

When compared to the embedded timer, SWBA, of the Atheros hardware, it can be noted that the hrtimer outperforms the SWBA timer in terms of precision. Should it be possible to adjust the firmware of the Atheros chips, then it could be that the SWBA timers could be more precise, since it would be able to trigger the interrupt routine directly. However, it might be that the coarser granularity of the bcntimer will keep providing issues regarding the performance. Kernel configurations such as dynticks, HZ, or preemption do not influence the performance of the hrtimer in a significant way. It needs to be stressed however that a Linux system is not a hard real time system and there are no guarantees that a timer interrupt will occur in time, as can be seen from the results, where once every 5 million times a couple of timer interrupts are skipped, due to some yet undetermined system activity.

Since the lack of support for hard real-time deadlines in the Linux kernel, the available timers of a number of real-time extensions or real-time operating systems is investigated. The RT-patch Linux kernel extension does not ensure a real-time operating systems, although it attempts to add real-time properties to the Linux kernel. The modifications of the patch ensure a completely different behavior of the Linux system, which also showed in the performance results, that indicated a worse hrtimer performance during a considerable load due to concurrent timers. The Xenomai real-time Linux kernel patch ensures real-time behavior for the applications and kernel modules that make use of the Xenomai real-time core interfaces. A performance evaluation of the real-time timer showed results even better than the hrtimer. However, the most interesting property of the Xenomai real-time timer is its determinism. This timer does not show any outlier measurement values, unlike the Linux kernel hrtimer. A third system was investigated, eCos, which is a dedicated RTOS. The performance of the eCos timer approaches the hardware limitations, showing an incredible performance. Note that the eCos system does not provide support for wireless NICs and comprises only a single application, targeted to a single purpose, while the Linux system is a multi-user, multi-threaded and multi-application system.

The second task of this chapter was to create a reliable slotted transmission mechanism based on IEEE Std 802.11n compliant COTS hardware. The hostapd application was transformed into a kernel module to eliminate any possible timing limitations on the association method. The ath9k driver was adjusted such that packets from higher layers are buffered until the timer source provides a trigger. The transmission of those packets is triggered by the scheduled timer by processing the packets in a FIFO manner. A considerable amount of the standard IEEE Std 802.11 operational functionality was disabled or circumvented, such as the disabling of Block Acks, power saving, ACKs, carrier sense, etc. The backoff window was minimized, all data and control packets use a single queue, the reception of packets was disabled after the association phase, etc. With all these measures in place, the test setup generated data packets as fast as possible, while network dumps were taken at the receiving side. From those dump files could be deduced that a quite reliable slotted transmission system was achieved, where only 0.09% of the received packets exhibit a deviation larger than 10 µs from the expected arrival time. Even in a high contention medium, the deviation remains bounded. However, there is no guarantee, since there are occasions where the data was received a whole slot too late.

In general, the results seem to be satisfying, a reliable TDMA system can be worked out on commodity hardware, however, care should be taken to construct a flexible protocol, where the slot boundaries are flexible and which is able to cope with data that is sent a slot too late. There are no hard guarantees that data will arrive on time.

Part III

Cognitive Radio Networks

CHAPTER 8

Rendezvous in Cognitive Radio Networks

8.1 Introduction

One of the primary drivers of Cognitive Radio (CR) research was the observation that the wireless spectrum is underutilized. The focus of CR research was already from the early start directed towards dynamic spectrum access and secondary use of the spectrum [224]. Note that CR is broader than just spectrum access, according to Mitola [225] CR is about having a flexible physical layer, a software defined radio, which is controlled by model-based reasoning. Others have another definition for CR, where the focus is placed on the flexibility of the transmission parameters based on the perceptions of the environment. A common element which most agree upon is the need to be flexible, that is, reconfigurable, and the radio should be sufficiently intelligent to deduce the need to reconfigure itself, that is, it should be able to learn and not just follow some algorithm. This is called the cognition cycle, which is an important aspect of CR, since it elevates CR from just an intelligent device. The cognition cycle incorporates the capturing of sensory stimuli and radio spectrum measurements, the analysis of these inputs, the learning and interpreting of those results and the acting upon those results.

The concept in which a Cognitive Radio Network (CRN) is situated used in this and the following chapter consists of the notion that a primary user (PU), which has been assigned a band of frequencies to be used during a certain time period, does not require the full set of frequencies 100 percent of the time. Therefore, secondary users (SUs) may try to access the medium through the usage of the unused frequency band, i.e. they have opportunistic access to these bands. Note that upon the appearance of the PU, a certain action needs to be undertaken, based upon the field of Cognitive Radio that is applicable. Some fields require the SUs to vacate the band immediately in order not to disturb the PU's communication, while others find it satisfactory if the SUs use a reduced power scheme such that the possibility to disturb the PU is minimized. Such type of network is characterized by a spectrum availability which changes over time and location. Secondary users therefore need to be agile and able to adapt themselves to changing spectrum opportunities. As such, The SUs are required to regularly sense the medium to capture an image of possible PU activity, possibly aided by information that is available from Radio Environmental Maps (REM). Due to the dynamic nature of the network, central coordination of the SUs is avoided, demanding of the nodes to exchange information in a distributed manner [226]. Note that centralized coordination is proposed in for example the IEEE Std 802.22 [227]. However, this centralized access includes coordination of the spectrum of the Primary User and therefore requires the assistance of the PU network.

Since cognitive radio is inherently a multi-channel network, specific MAC protocols are required. The multi-channel MAC protocols can be cataloged into several different approaches, upon which most related works have a similar view [228][229][230]: Dedicated Common Control Channel (CCC), Common Hopping, Split phase and Parallel Rendezvous.

The Dedicated Common Control Channel requires cognitive radio nodes to occupy at least two radio interfaces. One of those interfaces is used to exchange control messages over a dedicated channel, hence termed Dedicated Common Control Channel. The other radio interface is applied to exchange data over any channel. Such approach is beneficial since all nodes can overhear all control information of other nodes, thereby ensuring either a local or global coordination. In an opportunistic network, such as a Cognitive Radio Network (CRN), the allocation of such CCC might be an issue. The PU might appear at the channel used for control messages, requiring all SUs to maintain radio silence over this channel. Since the control channel is out of limits for the SUs, no agreement regarding a new control channel is possible.

The Split Phase approach is a variation of the CCC approach where nodes only have a single radio interface. The time is split into a control phase and a data phase. During the control phase, messages are exchanged over the common control channel, while data is exchanged over the assigned channels during the data phase. The improvement over CCC is the elimination of the requirement of two radio interfaces. However, the issue regarding the dedicated control channel remains and the approach results in a reduction of available time to transmit data. Moreover, time synchronization is required for the control slots and data slots to overlap between nodes.

A common channel hopping sequence is defined in the Common Hopping approach, according to which all nodes hop. The nodes only need a single radio interface which regularly switches to the next channel in the hopping sequence. Nodes that are willing to exchange data stay on the current channel and rejoin the hopping sequence when the data exchange is finalized. Unfortunately, this approach requires a strict synchronization between the nodes.

The Parallel RDV approach also requires just a single radio interface and is also based on a channel hopping sequence. The main difference is that this approach allows agreements between multiple nodes on distinct channels, thereby alleviating the stress on the single channel. A possible methodology to achieve this is for each idle node to follow a specific hopping sequence, while a possible transmitter is waiting on a certain channel. When applied correctly, that is, when no synchronization is required, this approach shows promising possibilities, as will be discussed later on. The search for the most optimal method has formed itself into an area of Cognitive Radio Network research, which is called Neighborhood Discovery (ND) or rendezvous and is focused on the search for an available channel through which two SUs can communicate.

This chapter and the next discusses Neighborhood discovery algorithms and their performance. Since some of the related work makes use of interdisciplinary concepts, the next sections go into detail of the definitions and methods. The related work is described in Section 8.4. Note that this chapter does not contain new results, it is included in order to understand the problem regarding a SU accessing the spectrum in a CRN. Moreover does it serve as an overview of existing approaches.

8.2 Channel Hopping based Rendezvous

There are multiple examples of channel hopping based rendezvous, of which just a limited subset is discussed in this section. The first example is Blind RDV, where the next channel is randomly selected by each node individually. Such channel assignment does not provide any form of guarantee, it is not even possible to give an indication of an absolute maximum for the time to rendezvous.

The Slotted Seeded Channel Hopping (SSCH) [231] is given as the next example. Each node makes use of a channel-seed pair to determine the channel sequence. $N-1$ seeds are available for N channels. Nodes with the same seed are guaranteed to have a rendezvous. The issue with this system is the requirement of having a synchronized subsystem. All slot boundaries need to be synchronized in order for the system to achieve rendezvous.

One of the most well known protocols that employs frequency hopping in order to achieve neighborhood discovery is Bluetooth, where each device, part of a piconet, makes use of a hopping sequence based upon the Bluetooth device address and clock of the master. The connection establishment procedure, paging, is initiated by the master by repeatedly transmitting the paging message in different hop channels. Since the Bluetooth clocks of the master and the slave are not synchronized, the master does not know exactly when the slave wakes up and on which hop frequency.

Therefore, it transmits a train of identical page scan messages at different hop frequencies and listens in between the transmit intervals until it receives a response from the slave. Since the paging message is a very short packet, the hop rate is 3200 hops/s. A device that is prepared to accept connections listens for page requests on the page physical channel, which follows a slower hopping pattern than the basic piconet physical channel and is a short pseudo-random hopping sequence through the RF channels. By having the master hop over different channels, according to a channel hopping sequence based upon the destination Bluetooth address, at a faster pace than the channel hopping of the destination, it will eventually accept a valid page message from the master.

In recent works, such as [232], [233], [234], [235] and [236], a new method has been designed to construct hopping sequences that provide a guaranteed rendezvous, even in an asynchronous environment. Such works make use of designs with specific mathematical properties, of which the next section lists some necessary definitions to elaborate on the related works.

8.3 Definitions

8.3.1 Quorum Systems

A Quorum was originally defined and used for a majority voting scheme in a distributed computing environment [237]. However, they also became common in areas such as replication protocols, operating systems, distributed mutual exclusion, power saving and other applications.

A common agreed upon definition can be found in [238], which defines a Quorum System as follows:

Definition 8.1 (Quorum System)**.** : Given a finite universal set $U = \{0, 1, \ldots, n-1\}$ with $n \geq 1$, a Quorum System Q under U is a collection of non-empty subsets of U, called quorums, satisfying the intersection property:
$\forall A, B \in Q : A \cap B \neq \emptyset$

Note that the Intersection Property is valid only for cycle and slot aligned systems, that is, the slots should be synchronized. Therefore, the QS Intersection Property is not sufficient in an asynchronous environment. In order to have a RDV guarantee in such case, a quorum must satisfy the Rotation Closure Property (RCP), which is defined as follows:

Definition 8.2 (Rotation Closure Property)**.** : For a quorum R in a quorum system Q under an universal set $U = \{0, ..., N-1\}$ and $i \in \{1, 2, ..., N-1\}$, there is defined: $rotate(R, i) = (x + i)\ mod\ N | x \in R$. A quorum system Q has the Rotation Closure Property if and only if
$\forall R', R \in Q, R' \cap rotate(R, i) \neq \emptyset$ for all $i \in \{1, 2, ..., N-1\}$.

Quorum Systems is a common denominator for several different types of quorums. Two of those types, a Grid Quorum System and a Torus Quorum System, are depicted in the next subsection, since they are used in the related work that needs to be discussed.

Grid Quorum System

In [239], a Quorum System was used to solve a mutual exclusion problem. All processes that have access to a shared resource need to ask permission to a subset of those processes, a quorum. If all processes in the quorum grant access, the resource is temporarily assigned to the requesting process. In order to achieve such feat, a new type of quorum was proposed, that is, a Grid Quorum System. Grid Quorum Systems are very popular Quorum Systems in amongst others power-saving protocols.

A Grid Quorum System is constructed by means of a square matrix, also called a grid. The number of elements contained by the grid is equal to $|S| = k \times k$, for some integer k. The set of subsets $Q_{i,j}$ of S forms the Quorum System over S, where each $Q_{i,j}$ contains all elements from row i and column j, with $1 \leq i \leq k$ and $1 \leq j \leq k$.

From the construction method can be deduced that such a quorum system has k^2 = N quorums, each of size $2k - 1$. For the mutual exclusion problem, such a Quorum System provides an efficient solution, since only $\sqrt{N}$ processes or nodes need to grant access for the requesting process or node to have exclusive access to the shared resource. In terms of Neighborhood Discovery, an interesting property is that every quorum intersects with every other quorum in at least 2 elements.

Moreover, the Rotation Closure Property is valid for Grid Quorum Systems, under the condition that the grid arrangement is constructed according to certain rules. Such grid allocation rules have been defined in [240], where a formal grid is formed. A quorum system, which satisfies the Rotation Closure Property (RCP), ensures that every quorum intersects at least once with every other quorum in an asynchronous operation.

Torus Quorum System

The Torus Quorum System has been designed by [241], based upon the Grid Quorum System. Instead of arranging nodes according a grid, nodes are arranged according to a rectangular array (wraparound mesh), called a torus. The last row (column) is followed by the first row (column) in a wraparound manner. The number of elements (N) that the array is composed of is determined by the height of the array (r), that is, the number of rows, and the width of the array (s), that is, the number of columns, where $N = r \times s$, with the constraint that $s \geq r \geq 1$.

A Torus Quorum System is composed of the set of quorums that are constructed according to some predefined constraints. Each quorum is of equal length and is composed of $r + \lfloor \frac{s}{2} \rfloor$ elements. From a $r \times s$ torus array, a column c_j $(j = 1 \ldots s)$ of r elements is included into the quorum. The remaining quorum elements are selected by picking one element from each of the $\lfloor \frac{s}{2} \rfloor$ succeeding columns. The selection of a single element from the columns is done in a wraparound manner. The group of elements that constitutes the single column c_j is called the quorum's head. The remainder of the elements form the tail of the quorum.

A more flexible construction method is proposed in [236], where a mirror torus extension constructs the tail of a Torus Quorum by selecting in total $\lfloor \frac{s}{2} \rfloor$ elements in a wraparound manner, on element from each column c_{j+k_i*i}, where $k_i \in \{1, -1\}$ and $i = 1..\lfloor \frac{s}{2} \rfloor$. Toruses of the same Torus Quorum System are required to select the elements from the same columns.

When employed in the context of distributed mutual exclusion and agreement protocols, the usage of a Torus Quorum System allows processes or nodes to reach an agreement with fewer required grants than the Grid Quorum System. The quorum of a Torus Quorum System in its most optimized configuration is of size $\sqrt{2N}$, whereas the size of a quorum in a Grid Quorum System is $\sqrt{N}$.

Like the Grid Quorum System, does the Torus Quorum System satisfy the Rotation Closure Property (RCP) requirements and can therefore be used in an asynchronous environment while still maintaining the intersection guarantee.

8.3.2 Cyclic Difference Set

The Cyclic Difference Set (CDS) [242] is closely related to a Quorum System, since the Cyclic Quorum System, proposed in [243], is based upon it. Moreover, the CDS satisfies the RCP condition and can therefore be employed in asynchronous environments. Other than in the Cyclic Quorum System, the CDS is very popular in power saving algorithms. The formal definition of a CDS is as follows:

Definition 8.3 (Cyclic Difference Set (CDS))**.** : A set D: $\{a_1, \ldots a_k\}$ $moduloN$, with $a_i \in 0, \ldots N-1$, is called a cyclic (N, k, λ)-difference set if for every $d \neq 0(modN)$ there are exactly λ ordered pairs (a_i, a_j), $a_i, a_j \in D$ such that $a_i - a_j \equiv d(modN)$.

As an example, the set {0,1,2,4,5,8,10} modulo 15 is a cyclic (15,7,3)-difference set. Besides the strict CDS definition, there exists also a relaxed CDS, where not exactly λ ordered pairs need to be found. Just a single ordered pair is already sufficient. The definition of the relaxed CDS is as follows:

Definition 8.4 (Relaxed Cyclic Difference Set)**.** : A set D: $\{a_1, \ldots a_k\}$ modulo N, $a_i \in 0, \ldots N-1$, is called a relaxed (cyclic) (N, k)-difference set if for every $d \neq 0(modN)$ there exists at least one ordered pair (a_i, a_j), $a_i, a_j \in D$ such that $a_i - a_j \equiv d(modN)$.

8.3.3 Latin Square

Latin Squares were introduced in 1783 by L. Euler as a "nouveau espèce de carrés magiques", a new kind of magic squares [244]. He pondered on the feasibility of arranging 36 officers. The officers were equally distributed between six different ranks and six different regiments. The arrangement of the officers should be in a square formation 6 by 6, such that there is only a single officer from each rank and regiment in each row and each column. A Latin Square of order n is therefore an $n \times n$ array of n symbols in which every symbol occurs exactly once in each row and column of the array.

Those contemplations led to the development of a branch of mathematics, which has become applicable into several fields, such as experimental designs (BIB - balanced incomplete block - designs), scheduling round-robin tournaments, Graph Theory, job assignment and processor scheduling for massive parallel computer systems.

8.3.4 Identical Row Square

Identical-Row Square (IRS) is an array which complies to the following definition according to [245]:

Definition 8.5 (Identical-Row Square (IRS))**.** : An Identical-Row Square (IRS) is an $n \times n$ table filled with n different numbers in such a way that each row consists of a permutation of integers in Z_n and all rows are identical.

8.4 Related Neighborhood Discovery Protocols

This section targets the related work regarding Neighborhood Discovery, also called Rendezvous, protocols in Cognitive Radio Networks. The related work is limited to Rendezvous protocols that make use of frequency hopping techniques. Works that rely on the functionality of a common control channel are not considered due to their weaknesses, such as single point of failure and low scalability.

There exists a wide variety of Neighborhood Discovery protocols, which can be differentiated by either their targeted environment or their methodology. Environments can amongst others be divided by the role the nodes assume; for example one node has a sender role, while the other nodes are receivers [245]. Another criterion which characterizes environments is synchronism, whether the protocol requires that all nodes are clock synchronized [246] or the protocol is designed for an asynchronous operation [246][247]. The available number of channels to all nodes can vary; some protocols take into account that due to external factors some nodes may experience a different view on the available channels. This is usually called an asymmetrical channel view [248][249]. Related to such view is the channel quality; nodes may experience a different channel quality, which may include detected PU activity, signal to noise interference, number of lost packets, etc. Some protocols take this into account by assigning a certain channel priority [250][251].

Classification based upon the targeted environment is a non trivial assignment, since protocols are often not focused on a single environment parameter, but to a whole set, which makes the differentiation difficult. The classification based upon the employed methodology is more straightforward, it ranges from a (pseudo-)random method to most deterministic algorithms that employ techniques such as Quorum Systems, Difference Sets or Latin Squares.

Some of the Rendezvous protocols assume an environment in which all nodes are synchronized, such as [252, 253, 254, 255, 256, 231, 257]. Note that the term synchronization is used in different contexts. The one time, slot synchronization is required, such that the slot boundaries of the nodes match, while in other proposals a complete frame synchronization is required, where slot x of node A is required to overlap with slot x' of node B. However, for nodes to become synchronized and maintain their synchronization in a multi-channel operation, the synchronization protocol should determine a rendezvous between the different nodes at a certain channel in order to exchange synchronization information. Hence, the Rendezvous protocol that assumes a frame synchronized network is reduced to a scheduling problem, since the rendezvous has already been achieved by the synchronization protocol.

The following subsections make a distinction between the different Neighborhood Discovery protocols, based upon the employed methodology. Note that protocols that are designed for multi-channel networks are not necessarily suitable for opportunistic networks. An excellent example is the protocol proposed in [258] and [259], where a MAC protocol is designed to operate in a multi-channel environment based upon IEEE Std 802.11 technology. While the protocol employs cyclic quorums to ensure the Rendezvous, it still requires for all nodes to have knowledge of the identification codes of its one hop neighbors. Such information is usually not available in an opportunistic network.

8.4.1 Random based Rendezvous

Rendezvous protocols that make use of random frequency hopping sequences ensure that there is an equal probability to achieve a rendezvous for each channel. Moreover, the methodology is straightforward, which makes it easy to incorporate into MAC protocols. A more advanced algorithm, AMRCC, which makes use of random hopping sequences is proposed in [260][261]. The protocol performs periodic spectrum sensing in order to detect the presence of PUs. Since the spectral signature of a PU is indistinguishable from other SUs, a simple energy detection is not deemed sufficient, a feature detection is assumed to be present. Based upon this information, a channel ranking table is built, considering the probability of interference with PUs. The pseudo-random hopping sequence is constructed such that higher priority channels obtain more opportunities than low priority channels by taking the channel ranking table into account. The decrease in RDV opportunity can be either a linear function or a parabolic function. The design of this algorithm allows a higher probability to achieve rendezvous on a channel with low PU activity.

Unfortunately, although this type of algorithm does provide a statistical probability of having a rendezvous within a certain time, the Time To Rendezvous (TTR), and lowers the probability of encountering a PU transmission, it can not ensure any absolute upper boundary of the to be expected TTR. The protocols discussed in this section employ in some manner a certain randomness, while considering the predictability of the algorithm, such as providing an upper bound to the TTR.

The algorithm described in [253] is comprised of five different channel hopping sequence selections, ranging from a single sequence to a more intelligent selection such that an upper bound on the TTR can be provided. The protocol assumes that all SUs are synchronized by means of a common time reference, for which a GPS device is provided as an example. In such manner, all nodes follow the same hopping sequence in a time synchronized manner in case of the single hopping sequence. A more intelligent channel hopping sequence selection is also proposed, where the transmission pattern is selected randomly, while the receiver first performs energy detection after which it performs the decoding of the received frames. Such scheme can provide an upper bound to the TTR.

The algorithm proposed in [262] does not randomly select channels, however, a permutation of the N channels is randomly selected. the selected permutation is then repeated N times, with a single permutation interspersed between the repetitions. Such design allows for a periodic and fair protocol, that is, all channels have an equal opportunity to rendezvous. Moreover, thanks to the specific design, an upper bound on the TTR is defined, even while working in an asynchronous environment.

A protocol that is partially situated in this section and partially in the next section was designed in [263], where the Modular Clock (MC) and Modified Modular Clock (MMC) algorithm were proposed. Both algorithms are based upon prime number modular arithmetic. In MC, all nodes compose a list with the observed channels. The lowest prime number (p) greater than the number of observed channels (m) is determined by each of the SUs. The first channel is chosen randomly from the observed channel list. A rate, or step size, r is randomly selected and determines the following selected channels, by increasing the channel index by r modulo p. If after an interval of $2p$ time slots no rendezvous has been achieved, a different r value is randomly selected. An issue with this protocol is the lack of a guaranteed rendezvous when SUs have selected the same prime number, which is for example the case when both SUs have an equal amount of observed channels. Therefore, the improved MMC protocol allows the random selection of a prime number between m and $2m$.

8.4.2 Number Theory based Rendezvous

Number theory is a research area within mathematics, specifically focused on the study of integers in the first place. The study of prime numbers and their characteristics can be categorized within this research area. Some interesting properties have been discovered when working with prime and co-prime numbers, which are employed in some of the following works. Note that the Modular Clock algorithm in [263] also used these properties, albeit the algorithm could not guarantee a rendezvous due to identical prime selection.

A similar approach has been taken to design the protocol described in [264]. In a system with N available channels, N channel hopping sequences are defined, which are used at random by the SUs. Each hopping sequence is of different length L_k, which should be greater than or equal to N and should be a prime number or a power of a prime number. This ensures that the length of two different sequences are co-prime and therefore guarantees rendezvous between any two SUs that select a different hopping sequence. In order to ensure rendezvous when the same hopping sequence is selected, a single channel k is arranged according to a quorum based algorithm in sequence S_k. The not yet allocated slots are assigned to the remainder of the channels. The approach allows for asynchronous rendezvous.

Two protocols are proposed in [265][246], a synchronous protocol and an asynchronous one, of which only the last is considered in this and the next chapter. The algorithm, ETCH, exploits characteristics of number theory based on prime numbers by applying addition modulo the prime number. A requirement is that the available number of channels (N) is prime. In total $N-1$ channel hopping sequences are generated, that are composed out of N frames. A frame consists of a pilot slot and a pair of subsequences that are derived by means of addition modulo N, where the subsequence index determines the channel step size. The combination of all pilot slot together forms the specific subsequence in the same order. Rendezvous is guaranteed in at least a single slot when SUs select the same channel hopping sequence and in N slots when different hopping sequences are selected. A disadvantage of the protocol is that it can only be used for a number of channels N that is prime. It could be argued [247] that the protocol could be extended to select channels in the same manner as in [263] and [266], such that it can be used for all possible number of channels. If N is not prime, the smallest prime P larger than N is chosen and a random channel from the available channel list is selected when the channel index is larger than N. Such modification, referred to as *mod_etch* in the next chapter, allows the usage of any number of channels at the cost of a decreased performance of the non-prime number of channels compared to the prime number of channels.

A notable algorithm that is partially based on Number Theory and partially on a jump-stay pattern is called JS [267][268]. The algorithm defines a single channel hopping sequence that consists of P frames composed out of three patterns of size P, with P being the smallest prime number greater than the number of channels N. Two of them are jump patterns during which the subsequent channel is selected based upon the index i and the step length r. The third pattern is a stay pattern, during which the SU stays on the same channel. During the jump pattern, the channel selection starts with index i and determines the next channel by means of step length r and modulo P operations. Every frame, that is, every $3P$ slots, the step length r is incremented by one while applying modulo N to it. Every N frames, that is, every $3NP$ time slots, the index i is incremented by one, while applying modulo P to it. The maximum TTR (MTTR) is proven to be $3P$ in the symmetric model, referred to as js_sym in the next chapter, where a homogeneous channel availability is assumed. The algorithm guarantees rendezvous even in an asynchronous environment.

An improved version of JS can be found in [249], where EJS is proposed. While JS supports a heterogeneous channel view, that is, every node can view a different number of available channels, both the size of the hopping cycle and the maximum TTR (MTTR) became very large for such environment. Compared with JS, the enhanced JS (EJS) lowers the upper-bounds of both the MTTR and the expected TTR from $O(P^3)$ to $O(P^2)$ under the asymmetric model, while keeping the same order $O(P)$ of upper-bounds of MTTR and E(TTR) under the symmetric mode.

The class of algorithms, Coordinated Channel Hopping (CCH), proposed in [269] acquired its inspiration from the JS protocol, although the protocol is focused on an asymmetric channel view, that is, every SU can observe a different list of available channels. Like the JS protocol, this algorithm makes use of a jump stage and a stay stage. The length of those stages is determined by a prime number P, which is the smallest prime number greater than the total number of available channels in the system, N. The length of the jump stage is $2P^2$ slots and the length of the stay stage is $2P$ slots. This cycle is repeated P times, during which the cycle index r, which determines the channel on which the stay stage is based, is incremented. In order to generate the channel hopping sequence during the jump stage, two lists are defined, an L_A list, which is the list of available channels for this SU, and an L_B list, which is a randomly ordered list containing values between 0 and $P - 1$. The jump stage is subdivided in P sub-stages, each of $2P$ slots length, where the channel selection is based upon the value of L_B at index i, the slot position within the sub-stage and the cycle index r. In order to match the channel indexes to the available channels, the obtained channel indexes from both the stay stage and the jump stage are matched to the channels within the list L_A. If the channel is present, the current slot is assigned to the specified channel, otherwise a channel is chosen randomly from the list L_A. The multi-user scenario makes use of the above described method, which is specifically targeted for a pair of SUs, while reducing the list of available channels to the common channels list with other SUs upon each RDV. It is claimed that this solution reduces the TTR with 80% in an environment with an heterogeneous channel view.

8.4.3 Logic Contemplation based Rendezvous

The protocols discussed in this section make use of pure logic into their design. For example, when two cars are driving in the opposite direction at the same circular lane, then they are bound to collide somewhere.

A well known protocol, ring-walk, with a logical design background is proposed in [270]. Each channel is represented by a vertex in a ring, such that generating a channel sequence is equivalent to visiting vertices in the ring. Each SU walks on the ring by visiting vertices with a certain velocity, determined by the unique ID of the SU, in the same direction as all other SUs. This can be either in a clockwise or counterclockwise direction. The speed of the SU determines which channel is the following channel in its channel hopping sequence. Since all channels are arranged in a ring, the SUs are eventually bound to meet each other. Unfortunately, since all users need to have a different speed, the TTR between SUs can vary and the number of users is bounded by the number of available channels. If the number of users is equal to the number of channels plus one, then $user_1$ and $user_{N+1}$ are hopping at the same relative speed modulo the number of channels and will therefore either always meet or never meet.

A protocol, QRCH, where also the rendezvous is guaranteed also by means of slower and faster hopping is defined in [233]. The channel assignment to slots is performed according to a Quorum System, however, the design defines a different hopping sequence for the sender and the receiver. The receiver is hopping at a lower rate than the sender. SUs are not assigned a specific role, both sequences are generated for a single node. If data needs to be transmitted, the transmission sequence is employed, the receiver sequence otherwise. While the methodology also employs a faster and slower hopping sequence, it alleviates all the disadvantages of [270]. Note that the distinction between transmission and reception could lead to a lower efficiency, compared to protocols that define a single hopping sequence for all states.

A notable protocol that also makes the distinction between sender and receiver, A-MOCH, is proposed in [245]. A-MOCH is an algorithm based on Latin Square (transmitter) and Identical Row Square (receiver) maps. The specific construction of the protocol results in a sender which hops on every channel during a single sub-cycle, while the receiver stays on a specific channel during a single sub-cycle, thereby giving the impression of a faster and slower hopping sequence. It is shown that with these two CH sequences in A-MOCH, the degree of overlapping is N in a cycle of N^2, that is, a single RDV on each channel per cycle. The maximum TTR is claimed to be equal to $N^2 - N + 1$. On the one hand, A-MOCH is very easy to construct, however, on the other hand, the biggest shortcoming of the protocol is the need of two different sequences (sender and receiver sequence) in order to ensure guaranteed RDV on all channels.

Like in the example given above, the observation was made in [271] that if two nodes rotate in different directions, the first clockwise and the other counterclockwise, they are guaranteed to meet. Based on this observation the protocol SARCH is developed, which is an asynchronous protocol. A sequence of channels in decreasing (N slots) and increasing (N slots) order is generated plus an additional slot on a specific channel, which is then repeated in a rotated manner $2 \times (2N+1)$ times, where N is the number of channels. The cycle size is equal to $2 \times (2N+1)^2$ and the MTTR the protocol enforces according to the authors is equal to $4N + 2$. A limitation of the protocol is that $2N + 1$ should be prime and therefore the protocol is not suitable for every N. Moreover, the construction of the protocol results in a rather large cycle size.

8.4.4 Guarantee System based Rendezvous

This section covers related work where the design aims at employing methods such as Quorum Systems, Difference Sets or Balanced Incomplete Block Design, which all guarantee RDV. Both the type of system as well as the manner of adoption can vary in the discussed protocols.

A Cyclic Quorum System is used in the design of QCH [232], where as many channel hopping sequences are generated as there are quorums in the Quorum System. A single channel hopping sequence consists of N frames, with N being the number of available channels. Each frame selects at random a channel from the available channel list and allocates the slots according to the quorum to this channel. The remainder of slots are allocated randomly to other channels. Since the Cyclic Quorum System satisfies the Rotation Closure Property (RCP), this algorithm is also suitable for an asynchronous environment.

A concept from Combinatorial Theory is used in [234] to guarantee rendezvous between SUs. Balanced Incomplete Block Design (BIBD), is defined as follows in the protocol description: a BIBD is an arrangement of v distinct objects into b blocks, such that each block contains exactly k distinct objects, with $k < v$. On the other hand, each object occurs in exactly r different blocks, and every pair of distinct objects a_i, a_j occurs together in exactly λ blocks. A property of BIBD guarantees the overlap between any two active blocks, that is, blocks that contain an object. By using this property, the Rendezvous protocol requires no synchronization, while still guaranteeing a rendezvous.

In [272] a protocol, ACH, is proposed that tries to minimize the risk of having a RDV failure due to the appearance of a PU. To achieve such feat, the number of RDV channels is maximized, a rendezvous on all available channels should be targeted within a single period. The protocol design makes use of an $N \times N$ array-based quorum, where sender and receiver obtain a different hopping sequence. While the construction allows for a RDV on all available channels, it requires a differentiation in the role of the SUs.

An improvement of the previous protocols is presented in [273], where also a symmetric algorithm, symmetric ACH, is proposed, that is, an algorithm that does not assume a sender or receiver role. The asymmetric channel hopping sequence design is based on an $M \times N$ array, where the one role makes use of columns and the other of rows. Such design ensures a larger number of distinct channel hopping sequences, compared to an $N \times N$ array. The symmetrical solution, where there is no sender or receiver role, is constructed by the concatenation of either the sender sequence or the receiver sequence, based upon the bit value within the unique ID, where each bit represents an asymmetric channel hopping sequence. The resulting cycle does still guarantee N rendezvous per cycle, however, the cycle is considerably larger compared to the asymmetrical case, thereby reducing the number of rendezvous per time unit.

In DSMMAC (Difference-Set-based asynchronous Multichannel MAC) [274] a single hopping sequence is defined for all SUs, based upon the concept of Difference Sets. The process of the forming of channel hopping sequences is not trivial; the (v, k, λ) Difference Sets [242] need to be selected in a very careful manner in order to ensure a high RDV probability. In DSMMAC, each channel is not necessarily mapped to the same number of slots, e.g., with 8 channels Channel 1 gets an extra slot, since all Difference Sets are chosen from a period of 73 slots, having 9 elements in a set. The remaining slot (slot 0) is assigned to Channel 1, and therefore SUs experience twice as much RDV on Channel 1 in a period.

MtQS-DSrdv (Mirror torus Quorum System and Difference Sets based rendezvous protocol) [275] is based on mirror torus QS and Difference Set concepts [236]. An equal distribution of the channels is targeted, that is, each channel is assigned an equal number of slots. Each CR determines its own hopping sequence which is constructed based partially upon a Mirror Torus Quorum System ($r \times s$, where r is number of rows/channels, and s the number of columns) and partially upon a more difficult to be constructed Difference Set. The protocol includes two different concepts in the construction of channel maps, thereby reducing the difficult Difference Set construction to a minimum. Both homogeneous and heterogeneous channel availability are considered, ensuring RDV on all channels in the former case, and on almost all channels in the latter case during a single cycle.

A protocol based upon a Mirror Torus Quorum System, mT-GQS, is proposed in [276]. The torus quorum is used in a different manner, since the torus quorum is selected from a grid array. The selection favors better quality channels by allocating more slots to them compared to worse quality channels.

The concept of Relaxed Difference Sets is employed in DRDS [266]. A hopping sequence is built based on Disjoint Relaxed Difference Sets (DRDS) guaranteeing RDV on each available channel in a single cycle. It is claimed that a maximum TTR of $3P$ can be achieved in the symmetric model, with P being a prime number greater than N, the number of channels. However, the reader must note that it is assumed that one SU is already in the network and another one joins the network. Before starting the Difference Set based hopping sequence, the just started node occupies for $2P$ slots on the available channel with the smallest index (e.g., Channel 1 with

3 channels). In this case, the MTTR estimation is correct. However, in the case that nodes have already started their hopping sequence and would like to meet other nodes, this estimation is false.

8.5 Conclusion

This chapter considered the opportunistic spectrum access of secondary users (SUs) in a cognitive radio network. Due to the dynamic properties of the network (CRN), the coordination of the SUs needs to be flexible and should not exhibit a single point of failure. Several methods are discussed, such as Dedicated Common Control Channel (CCC), Common Hopping, Split phase and Parallel Rendezvous. The first three methods exhibit considerable disadvantages, such as the requirement of having multiple radio interfaces, the need for synchronization, having a single point of failure, etc. On the other hand, the Parallel Rendezvous has interesting properties which can be exploited in the context of a CRN. One such characteristic is channel hopping, which alleviates the single point of failure problem, since the method does not depend on a single channel.

In order to understand the related work and its implications, some definitions and concepts, such as Quorum Systems, Cyclic Difference Sets, Latin Square, etc. are considered before discussing the related work. The related work can be categorized according to four classes upon which the Rendezvous is based: random, number theory, logic contemplation and guarantee system. The random based RDV schemes either provide an equal probability of having a RDV on a certain channel, or employ a channel ranking to ensure that a RDV is more likely to happen on high priority channels. Some of the methods also consider asynchronous environments. The RDV methods based on number theory is mostly based on the characteristics of prime and co-prime numbers. As such a RDV is guaranteed, however, the construction methods often need the number of channels to comply to certain requirements. Therefore, these methods are not suited for all possible number of channels. Logic contemplation is used when designing RDV methods that work according to the clockwise and counterclockwise method. It is logical that at some point in time the two will meet. The last class of protocols employ a system which guarantees a RDV, such as a quorum system. All those protocol guarantee a RDV on at least a single channel, possibly on multiple channels.

CHAPTER 9

Asynchronous operation in Cognitive Radio Networks

9.1 Introduction

This chapter covers joint work with Dr. S.A. Romaszko, where she devised the largest part of the Neighborhood Discovery protocol details, while our work was focused on the implementation and evaluation of the asynchronism, which can be found in Sections 9.2.2, 9.2.3 and 9.3.2. An improvement of the asymetric asynchronous extension is also a contribution of this work, depicted in Section 9.4. The focus is placed on SUs in the presence of PUs, where no central spectrum allocation management is available. Each CR user is equipped with a single tunable half-duplex radio transceiver which can switch between N different channels. A neighborhood discovery phase relates to a rendezvous on the same channel within a certain minimum duration between two SUs that consider a channel hopping sequence in order to establish a communication. In this chapter it is assumed that SUs are viable to procure a list of available channels. The spectrum holes, which an SU should identify, can vary in time and space. Based on a spectrum detection approach (sensing, database) each user recognizes a list of spectrum holes that can be used while respecting the PU's activity. It is assumed that channels are slowly time-varying, the system is slowly dynamic.

As already mentioned in the previous chapter, the assumption of synchronization in a CRN is similar to requiring the synchronization protocol to perform its own Neighborhood Discovery. Such practice is inefficient due to the double rendezvous requirement and reduces the following Rendezvous protocol to a mere scheduling problem. Therefore, this chapter assumes that nodes are working in an asynchronous manner in a distributed CRN. In a few works it has been claimed and verified that the asynchronous environment can even increase the performance of the Rendezvous protocol. An asynchronous comparison between the algorithms described in [263] and [253] is performed in [277], by implementing both on a testbed using Universal Software Radio Peripheral (USRP). The comparison showed that the experienced asynchronism reduced the Time to Rendezvous (TTR).

It should be noted that an asynchronous environment is the natural state in which the network is situated. In order to enforce a synchronous behavior, actions need to be taken, such that nodes operate in a synchronized manner. Making use of the natural state of a network provides already certain benefits compared to an enforced state. Moreover, in an asynchronous environment, two nodes might meet each other sooner, i.e., the Time to Rendezvous is shorter, thanks to the partial slot overlap with two slots of the second node, whereas in a synchronized environment, the slots overlap only with a single slot of the second node. However, a thorough analysis is required, in order to determine whether the slot overlap is sufficient in order for a node to contact its neighbors, or to determine whether the slot overlap is too low for any viable communication transmission.

Some of the discussed protocols in Chapter 7 do not take into account the asynchronous character of a CRN or propose a trivial addendum which, though it guarantees a RDV, has a very low efficiency. Most of the discussed protocols do consider the asynchronism and provide theoretical contemplations mostly regarding the maximum TTR, however the effect of the asynchronism is rarely studied. Often the medium is assumed to be perfect, since the RDV is expected to happen at the first possible occurrence, that is, when one SU is already hopping according to its hopping sequence, while the second SU starts its hopping sequence. This is a faulty assumption since, amongst others, interference can cause the loss of a packet or the false reception of a few bits can cause the complete packet to be discarded due to a false CRC checksum. Therefore, the RDV does not need to occur at the first opportunity, it could happen that both SUs are hopping according to their hopping sequence and find each other at some other moment. This chapter therefore evaluates the specific protocols for each possible asynchronous offset, which is expressed in percentage of a slot when all slots are of equal size. When a protocol employs slots of different sizes, the asynchronous offset is expressed in percentage of the largest slot. A more realistic view can be shed upon the performance of the protocols by evaluating them for every possible offset and calculating the minimum, maximum, average and standard deviation of the measured parameters.

Few papers include the rendezvous method, that is, how CRs establish a communication with other SUs. Commonly used methods include transmission and reception of an RTS/CTS sequence and the broadcasting of beacons. When considering the RDV, none of the protocols consider the required time to facilitate this message exchange. It is often assumed that the overlap time, that is, the time two SUs are present at the same time at the same channel, is long enough to supply two message exchanges. Since the available time to rendezvous, the overlap time, is a critical factor to determine whether a rendezvous could succeed, the asynchronous threshold is defined in this chapter, which indicates the minimum percentage of slot overlap is required to enable a RDV. This parameter in combination with the asynchronous offset allows to study the asynchronous behavior of the RDV protocols in more detail and study the effects of the asynchronism.

The remainder of this chapter discusses a symmetric slot assignment on the one hand and an asymmetric slot assignment on the other hand. Section 9.2 covers the asynchronous analysis of a protocol developed by Dr. S.A. Romaszko and the comparison with related work in a symmetric slot assignment, that is, where the slots sizes remain constant. An asymmetric slot assignment does not necessarily assign an equal amount of time to each slot, depending on the allocated channel. The remainder of the sections discusses an asymmetric slot assignment. Section 9.3 discusses the asynchronous analysis of such a protocol, ARE, with an asymmetric slot assignment. Section 9.4 contemplates on the improvement of the ARE protocol, where new methods to engender asynchronism are defined, which are, in combination with EAND, compared to ARE and other related protocols. Both the asynchronous RDV extension (ARE) (Section 9.3.1), which results in an induced asynchronism when applied to known RDV protocols and the Asynchronous Neighborhood Discovery protocol (AND) (Section 9.4.1), specifically designed to be used in combination with the asynchronous extension, are developed by Dr. S.A. Romaszko.

9.2 Symmetric Asynchronous RDV Analysis

This section discusses the analysis of the asynchronous behavior of a protocol that ensures RDV by allocating equally sized slots to channels. The protocol takes into account the possibility of having a different channel view between SUs, that is, it considers a heterogeneous channel view. Moreover, the probability of having a RDV is dependent on the channel ranking, higher quality channels have a higher probability than lower quality channels. Therefore, each SU is responsible for creating its channel hopping sequence according to the rules specified in the next subsection. This process of building a hopping sequence is a fully distributed process where no knowledge from other SUs is required.

The proposed protocol makes use of the properties of a torus QS. A torus QS (tQS) adopts a rectangular array structure called torus, where the last row (column) is followed by the first row (column) in a wrap-around manner. The height of the torus is specified as r (number of rows) and the width of the torus is specified as s (number of columns), where $N = r \times s$ and $s \geq r \geq 1$. A torus quorum consists of $r + \lfloor \frac{s}{2} \rfloor$ elements, where r elements are determined by selecting a column. That group of elements is called the head of the torus quorum. The remainder of the elements is chosen from the $\lfloor \frac{s}{2} \rfloor$ succeeding columns. That group of elements is called the tail of the torus quorum. The discussed protocol employs a different approach for selecting the tail elements, such as proposed in [278]. A special adjustment is made to this protocol such that a different element arrangement is used from which both the head and tail elements are selected.

After discussing the protocol specifics, the asynchronous verification procedure is examined, where the distinction is made between different cases that are differentiated by the combination of asynchronous offset and threshold. Thereupon based, the performance of the discussed protocol is measured and compared to a related protocol without the special adjustments, as well as to a random channel allocation method.

9.2.1 Mirror Torus-in-Grid quorum system (mT-Gqs)

The mT-Gqs algorithm is based upon a torus Quorum System where the tail selection is done according to the mirror torus extension [278]. The extension allows to construct the tail in a more flexible manner, which therefore introduces a randomization factor. The mirror torus extension constructs a tail by selecting its elements in a wraparound manner from column $c_j + k_i \times i$ with c_j being the column of the head, $i \in \{1 \ldots \lfloor \frac{s}{2} \rfloor\}$ and $k_i \in \{-1, 1\}$. The elements are selected from either a forward column or a backward column, a single element per column and the quorums of the same Quorum System should all adhere to the same directions, that is, $\forall i \in \{1 \ldots \lfloor \frac{s}{2} \rfloor\} : k_i = k'_i$.

Specific to the mT-Gqs algorithm is the employment of a torus quorum construction, not from an $r \times s$ array, but from an $N \times N$ grid array, with N being the number of available channels. The rows and columns are matched to the channel indexes, that is, the elements for Channel 1 (C1) are primarily selected from Column 1 and Row 1. The channels are assigned in function of their relative quality. The best channel is assigned the most slots, the worst channel the fewest slots. In order to achieve this, the highest priority channel, i.e., the channels with the best quality, first selects the complete row that matches with its channel index. Those elements form the head of the quorum. From the column that matches the index of the channel are the $\lfloor \frac{N}{2} \rfloor$ tail elements selected according to the mirror torus extension in the vertical direction. The remaining elements from that column is assigned to the worst channels in such manner that no more slots are allocated to a lower priority channel in comparison with a higher priority channel.

The formerly discussed procedure allows the creation of the torus in grid quorum for the highest priority channel. SUs that have a similar channel view, i.e., they have the same channel with the best quality, will always meet thanks to the quorum intersection property. The slot assignment of the remaining channels is performed in a similar manner, however, the set of slots do not form a quorum. The following channel, according to the list ordered by channel quality, allocates the elements from the row that matches its index that are not already selected by a higher priority channel. The number of tail elements are calculated from the size of grid (N) minus the number of already processed channels (n). The $\lfloor \frac{N-n}{2} \rfloor$ tail elements are selected in a mirror torus extension manner from the column that matches the channel index. The remaining elements from that row are assigned to the worst channels, such that no more slots are assigned to lower priority channels in comparison with higher priority channels. This procedure continues until a 2×2 array of unassigned slots remains, which is assigned in a diagonal manner to the second lowest priority channel and the lowest priority channel.

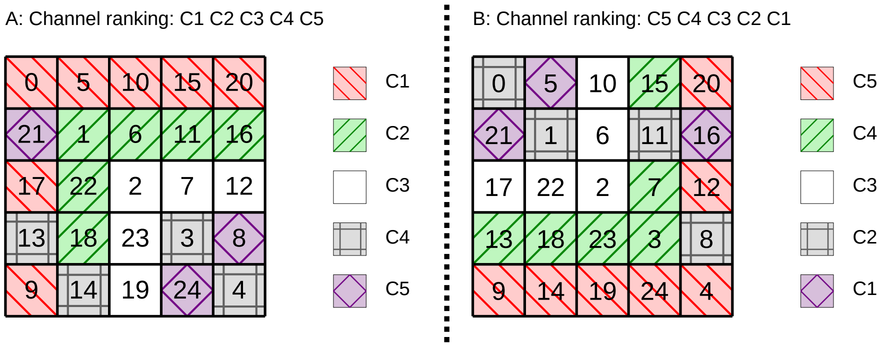


Figure 9.1: *mT-Gqs channel mapping*

Figure 9.1 depicts two examples of a slot assignment for five channels according to the algorithm. The slot assignment is shown for two channel orders, the second (B) being the reverse order of the first (A). The grid is constructed such that the diagonal distribution [236] of elements in an array is used. The slot numbers are allocated according to the positive diagonal rule, i.e., elements are ordered according to:

$$f(x, y) = ((y \times r) - ((r - 1) \times x)) modulo(r \times r) \tag{9.1}$$

where $x = 0, ..., N - 1$, $y = 0, ..., N - 1$, and $r = \sqrt{N}$. Such grid is proven to guarantee a RDV, even in an asynchronous environment, since the quorum complies to the RCP. A different method of allocating the slot numbers is the standard distribution, which assigns consecutive elements, row after row. Both methods are analyzed in the performance section.

9.2.2 Asynchronous analysis

The channel hopping sequence is considered to be periodic, of which the repetition rate is determined by the length of the sequence, and can therefore be analyzed by marking every slot that is assigned a matching channel in a synchronous analysis.

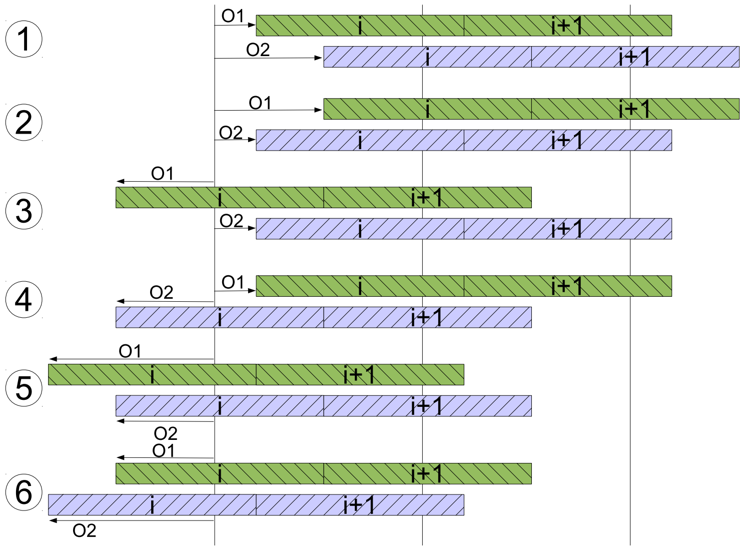


Figure 9.2: *asynchronous offset cases*

Two hopping sequences, where one is shifted with an offset x number of slots compared to the other, are compared to each other during a couple of periods and this for every $x \in 0 \dots M$, where M equals the length of the channel hopping sequence. From the gathered coincidence information, the TTR, MTTR, number of RDVs, etc., can be derived, as well as their distribution over the complete range of offsets.

The analysis for an asynchronous environment is more challenging, since the slot boundaries do not necessarily match. This section proposes a method which allows the asynchronous analysis of a RDV protocol in a manner as trivial as the synchronous analysis. To produce such method, the asynchronous offset A_o and the asynchronous threshold A_t is introduced. The former defines the deviation of the slot boundary compared to a synchronous slot boundary and is expressed as a fraction of a slot with $\frac{-99 \times S}{100} \leq A_o \leq \frac{99 \times S}{100}$, where S is the slot size. The asynchronous threshold defines the minimum fraction of a slot, expressed in percentage of a slot, that is required to consider the coincidence as a RDV. By including the asynchronous offset in the synchronous analysis and checking at every slot boundary from both sequences, the same method can be used to analyze the asynchronous behavior of a RDV protocol.

In order to prevent the need to repeat such comparison for every possible asynchronous offset, which is dependent on the required granularity of the measurement, the different possible positions of the asynchronous offsets are considered to reduce the needed effort. As a result of this analysis, six different cases can be discerned, which are depicted in Figure 9.2. Since all slots are of equal size, a slot of hopping sequence A can overlap with maximally two slots of hopping sequence B. The six cases that are depicted in the figure can be easily expressed by means of Equation

9.2 and Equation 9.3, where O_h is the highest offset, O_l is the lowest offset.

$$\begin{aligned} olap_{il} &= (O_h - O_l)mod(100) \\ slot_{il} &= i - (\lfloor (O_h - O_l)/100 \rfloor + 1) \end{aligned} \tag{9.2}$$

$$\begin{aligned} olap_{ih} &= 100 - (O_h - O_l)mod(100) \\ slot_{ih} &= i - \lfloor (O_h - O_l)/100 \rfloor \end{aligned} \tag{9.3}$$

The equations formulate the corresponding slot indexes of the second sequence ($slot_{il}$ and $slot_{ih}$) that have an overlap with $slot_i$ of the first sequence. The respective overlap sizes are formulated by $olap_{il}$ and $olap_{il}$. Based upon this information, nearly the same procedure as in the synchronous analysis needs to be performed only once for each possible offset x, that is, for every $x \in 0 \ldots M$, where M equals the length of the channel hopping sequence. The procedure deviates by the requirement to perform an analysis on both the slot boundaries of sequence A and sequence B, since both can occur at different instances. The asynchronism is verified by means of both equations, which are employed at every slot boundary. If one of the overlap values is greater than the predefined asynchronous threshold, A_t, a RDV is considered to be possible and therefore the coincidence between the two slots will be taken into account.

Note that in a synchronous analysis, the metric RDV is conceived as the number of slots a RDV opportunity is possible per cycle. Since asynchronism is considered in the performance analysis, care needs to be taken not to disregard the possibility of having the same channel in succeeding slots. Such occurrence would not result in a RDV in two different slots, but in a single RDV with a larger overlap time. To make the distinction, the term *RDV opportunity*, i.e., the number of rendezvous on sequentially distinct channels, is used in the performance section.

9.2.3 Performance evaluation

The mT-Gqs algorithm is compared with the slot allocation algorithm designed in [248], and a pseudo random method. The former maps channels to slots using a standard torus in grid QS, i.e., a tail is selected from succeeding columns in the same direction. This protocol is referred to as 'forward mapping' in this section, while mT-Gqs is referred to as 'mirror mapping'. The pseudo random method allocates slots in a random manner to channels, with the constraint of having the same channel distribution as mT-Gqs, i.e., the same number of slots are assigned to a certain channel priority. As a surplus, the diagonal (diag) and standard (std) slot arrangement in a grid is analyzed for both forward and mirror mapping.

The performance evaluation depicts both the best and worst case, that is, sequences that have the same channel priority list and the reverse channel priority list respectively. Under those circumstances the protocols are compared for 5, 10 and

20 channels. As already discussed in the previous sections, the measurements are performed for every slot offset within a cycle. The asynchronous threshold is always 30%, the asynchronous offset of sequence A is always equal to 0 and the asynchronous offset of sequence B complies to $A_o \in \{0\%, 30\%, 50\%, 70\%, 90\%\}$. The evaluation based on these parameters provides a complete overview of the performance of all three protocols for every possible offset, both synchronous and asynchronous.

Since the mt-Gqs algorithm and the pseudo random algorithm are composed of some random elements, the simulations have been run 1000 times for every possible configuration, resulting in a minimum, average, maximum and standard deviation for each metric.

The next subsection discussed the performance in a homogeneous environment, that is, all SUs experience a channel view with the same channels. Although all SUs have the same number of channels available, the channel ranking list might still be in a different order. The subsection after that discusses the performance in an environment where not all SUs have knowledge of the same list of available channels.

Homogeneous channel availability

This section targets the comparison between the forward and mirror mapping methods, where SUs are considered to have the same channels list. The performance is measured for both the diag and std grid distribution, in both the same and reverse channel order. The analyzed performance metrics include the Time To Rendezvous (TTR), the number of rendezvous opportunities, which denominate the number of rendezvous on sequentially distinct channels, and the total sum of the overlapping regions. The last metric is interesting in combination with the results from the RDV opportunity. Although the asynchronous environment is likely to have a lower total overlap time, it does not need to signify that the RDV opportunity is also lower. The asynchronism can result in more but smaller RDV opportunities. All figures depict the results by showing the 95% confidence interval for all slot offsets with the boxes, while the minimum and maximum value are shown by the lower end and upper end of the lines. Since the mirror mapping method includes some randomness in the protocol design, two boxes are shown that depict the maximum 95% confidence interval and the minimum 95% confidence interval. The black lines, shown within the boxes, depict the average value.

Figure 9.3 depicts the RDV opportunity for both forward and mirror mapping methods, with a grid distribution according to diag and std. The maximum value is not always shown to emphasize the more interesting values, since it is determined by the grid size, which is in direct relation to the number of channels, and is therefore a constant value per sub-figure.

The results clearly show the improvement of the RDV opportunity in the asynchronous environment over the synchronous environment. The asynchronous environment is represented by the asynchronous offsets 30, 50 and 70 percent, since the

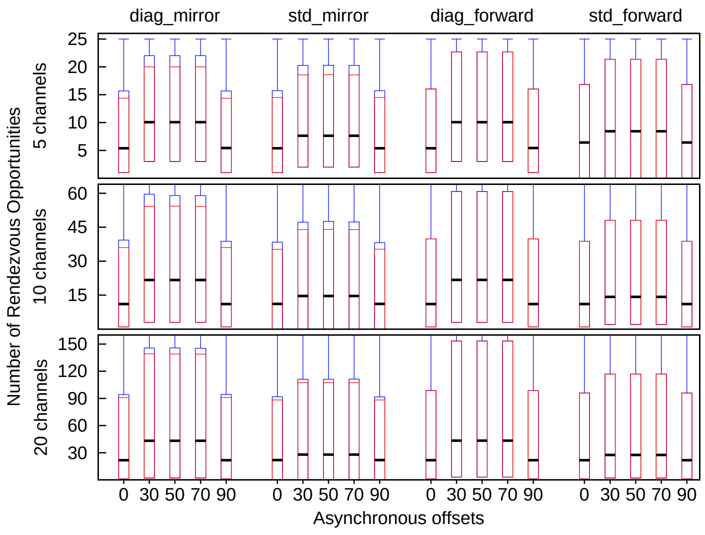


Figure 9.3: *RDV opportunity for forward and mirror in the same channel order*

asynchronous threshold is equal to 30%. Therefore, when an asynchronous offset (A_o) of 90% is verified, one of the overlaps is insufficient to support a RDV and provides therefore as much RDV opportunities as in the synchronous case ($A_o = 0\%$), although each overlap provides fewer time to perform the RDV.

Such behavior can also be found in the average overlap sum, depicted in Figure 9.4, where the statistics regarding the sum of the overlap are shown for the same measurement analysis. While, as expected, the 95% confidence interval of the synchronous measurement is larger than that of the asynchronous environment, is interesting to note that the average overlap per cycle of the asynchronous case is equal or nearly the same as the average overlap in the synchronous case. Nevertheless, even while the synchronous measurements has a larger 95% confidence interval, which suggests that more values are situated in the upper region, the difference between the overlap per cycle is small, although there is a significant difference in the RDV opportunities.

A similar behavior as the RDV opportunities can be found in Figure 9.5, which depicts the TTR (expressed in the number of slots) for a channel priority list in the same order. This supports the theory that an asynchronous environment results in more but smaller RDV opportunities, even while according to the total overlap sum, the total available Time To RDV decreases only a little.

When studying the different protocols and the grid distributions, it is also clear that the mirror mapping method provides some improvements over the forward mapping method. The randomness in the allocation of slots for the mirror mapping can be found back in the figures, where it is shown to exhibit a somewhat lower RDV opportunity. However, the average RDV values remain similar and the maximum

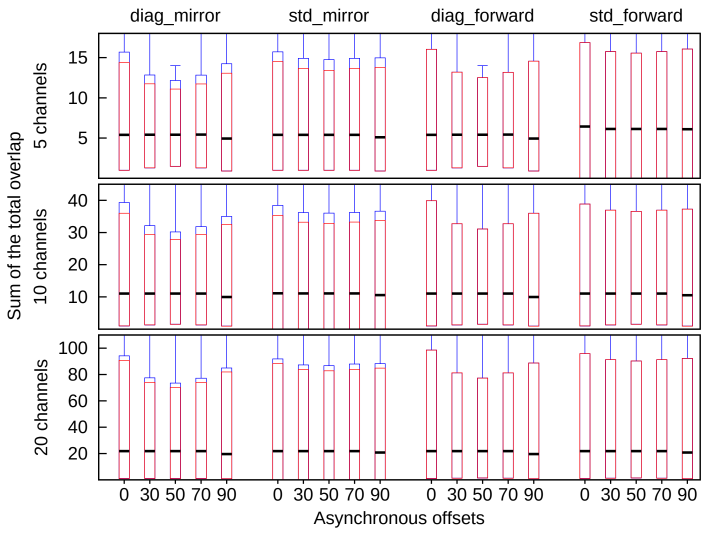


Figure 9.4: *Overlap sum for forward and mirror maps in the same channel order*

confidence intervals of TTR are lowered with a reasonable margin.

It is also clear that the use of a diagonal distribution results in a better RDV opportunity performance compared to the standard distribution for both mapping methods, and a better TTR performance with the mirror mapping. While applying diag with forward mapping the TTR averages are still better than that with std, however, the maximum confidence interval of TTR is worse.

All formerly discussed results are performed under the assumption that the channel ordering is the same, i.e., the best possible case. The worst possible case is when the highest priority channel of one SU is the lowest priority channel of the other SU. In such setup, the number of RDV opportunities are expected to drop, which is also the case, the average number of RDV opportunities drops from ~ 10 to ~ 7.5 for the mirror mapping method with a diagonal grid distribution. However, the relative values between the compared protocols and distributions are similar as in the same channel ordering, which is also the reason these results are not shown here. Instead, the TTR results are shown in Figure 9.6, where a clear distinction can be made between the different cases.

The benefit of using the asynchronous characteristics is clearly present in the figure for all cases. Likewise, the diag grid distribution provides better TTR results compared to the std distribution for both mapping methods. Like in the case with the same channel ordering, there is no significant difference between the forward and mirror mapping method, except that the mirror mapping method allows for some randomness and could therefore provide a lower maximum confidence interval compared to the forward mapping method.

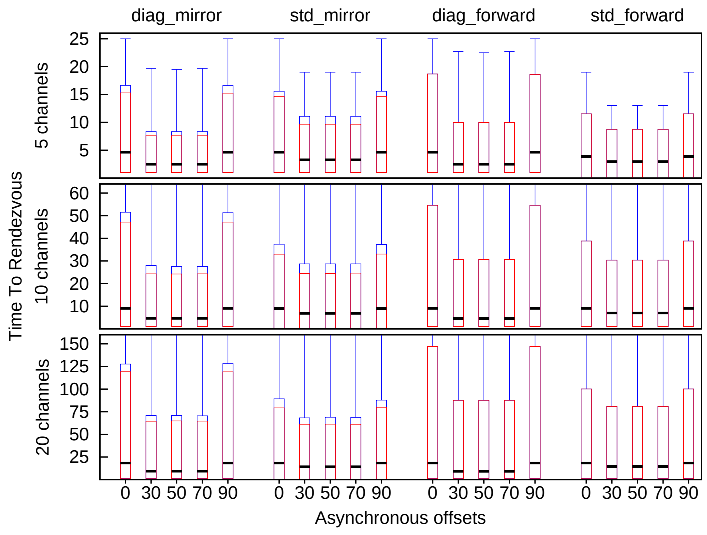


Figure 9.5: *TTR for forward and mirror maps in the same channel order*

Heterogeneous channel availability

This section discusses the comparison between mT-Gqs and the pseudo random method, where SUs are considered to have a deviating channel view. Both the similar channel order and the reverse channel order are considered while performing the analysis. Since the SUs have a different channel view, the same channel order is not possible, therefore a similar channel order is considered, where the common channels maintain the same order. The performance is measured in terms of RDV and TTR.

In the first considered case SU A has five channels, while SU B has seven channels. In the second case SU A has ten channels, while SU B has five channels. The obtained results are shown in Table 9.1, where the following metrics are depicted: minimum number of RDVs (minRDV), RDV opportunity (RDVopp), average TTR (TTR) and maximum TTR (MTTR). The channel ordering is specified by either being similar or reverse. The 'Asyn' column specifies whether the analysis was performed based on a synchronous or asynchronous environment.

From the results it is clear that in both synchronous and asynchronous environments mT-Gqs outperforms the random method. The minimum RDV values of mT-Gqs are much higher, while the maximum TTR values are significantly lower. The results justify the usage of a systematic approach over a random one.

Figure 9.7 shows in more detail the RDV opportunity for the 5-7 channels combination, with a 30% asynchronous threshold. The average values when using mT-Gqs

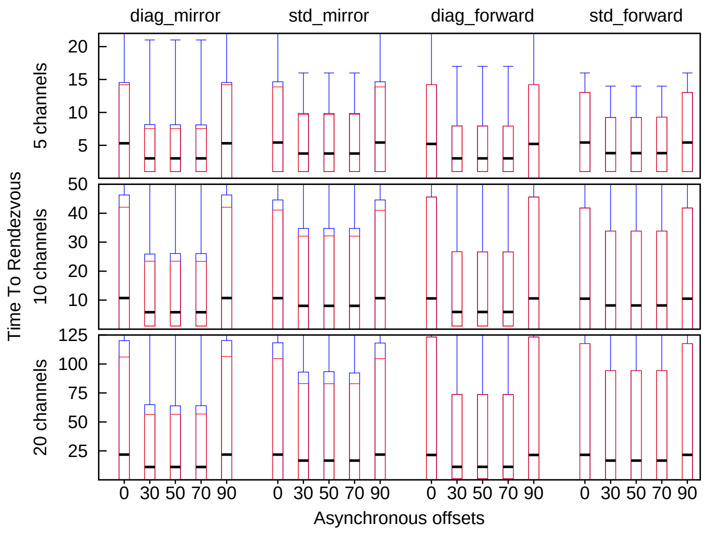


Figure 9.6: *TTR for forward and mirror maps in the reverse channel order*

Table 9.1: *asynchronous evaluation in a heterogeneous network*

Alg	Ch order	Chnls	Asyn	RDVopp	minRDV	TTR	MTTR
mT-Gqs	similar	5-7	SYN	8.28	4	5.92	35
random	similar	5-7	SYN	8.28	0	5.92	65
mT-Gqs	reverse	5-7	SYN	7.32	3	6.69	41
random	reverse	5-7	SYN	7.32	0	6.96	71
mT-Gqs	similar	5-7	ASYN	15.68	10	3.13	17
random	similar	5-7	ASYN	14.11	3	3.48	38
mT-Gqs	reverse	5-7	ASYN	13.60	9	3.60	18
random	reverse	5-7	ASYN	12.65	1	3.88	39
mT-Gqs	similar	10-5	ASYN	25.52	13	3.92	43
random	similar	10-5	ASYN	23.18	10	4.32	50
mT-Gqs	reverse	10-5	ASYN	23.52	14	4.25	35
random	reverse	10-5	ASYN	21.95	10	4.56	53

are clearly higher when compared with a random distribution for channels with a similar channel order when operating in an asynchronous manner. The synchronous operation leads to a similar average number of RDV opportunities for both mT-Gqs and the random distribution, which can be explained because of the averaging of the RDV results for every slot offset, combined with the usage of a random distribution with as many channels per priority as mT-Gqs and which is mapped alike. However, the maximum number of RDV opportunities is almost twice as much on the highest priority channel when using mT-Gqs, compared to random (respectively 15 slots and 8 slots), while on the lowest priority channel there still is a considerable

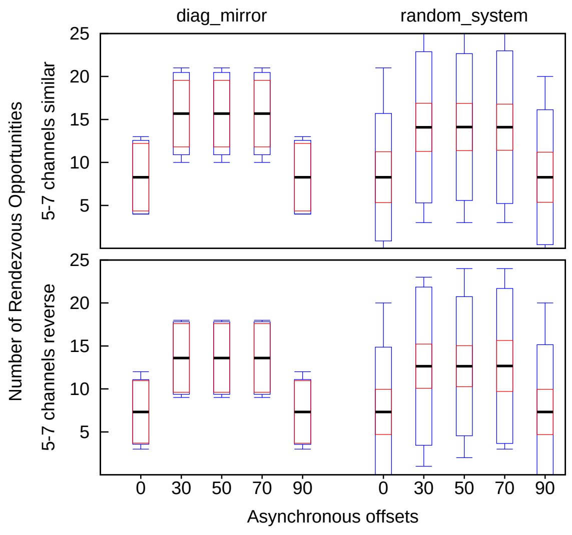


Figure 9.7: *RDV for mirror and random with asymmetric channel view*

difference (respectively 6 slots and 4 slots).

It is also clear that the standard deviation, or the variation on the standard deviation (the difference between the red and blue boxes), of the random distribution is far greater than that of mT-Gqs. This indicates that the random distribution is not as stable as mT-Gqs, which is to be expected. Moreover, the proposed protocol guarantees a RDV in every cycle, while the random distribution can result into cycles without any RDV. Similar results can be seen when both maps use the reverse channel order.

To summarize, this section discussed a method to analyze Rendezvous protocols with fixed slot sizes in an asynchronous manner without generating too much overhead. The method makes use of an asynchronous offset and an asynchronous threshold. The method is employed to analyze the behavior of a newly devised protocol that is based upon torus in grid quorums. The results highlight the already expected performance improvement thanks to the asynchronism. The RDV opportunity increases and the TTR decreases significantly when the asynchronous environment is compared with the synchronous environment. It is also shown that the proposed algorithm is not only much more stable compared to the random method, but it also improves the minimum RDV and maximum TTR performances.

9.3 Asymmetric Asynchronous RDV Analysis

The previous section discussed the asynchronous analysis of protocols with an equal distribution of the size of slots, that is, all slot were of equal size. This and the next sections contemplate on the asynchronous analysis of protocols where the size of the slot is considered to be variable. All nodes are assumed to be capable of identifying the available spectrum holes and composing a list of available channels, ordered according to their QoS. The quality of the channels, which determines the priority of the channels, can be measured by means of several parameters, for example the number of occurrences of a PU. The slot size varies with the priority of the assigned channel, where higher priority channels obtain a higher probability of having a RDV, since they have a larger slot and therefore more time to have a RDV.

Since the slot sizes are variable, there is no trivial rule to determine the asynchronous overlap between two sequences at a certain offset. In order to define both the slot size and the overlap with a certain precision, the slot size is divided into microslots. Each microslot has the size of one percent of the maximum slot size. A full slot, also referred to as a slot of maximum slot size, is therefore composed out of 100 microslots. Therefore, the slot size, and also the overlap, can be expressed in percents of the maximum slot, e.g. a slot can have a size of 70% of the maximum slot size. Since the size of a microslot is in direct relation to its duration, a discrete time line is created with a maximum resolution of one percent. By verifying the channel of both sequences at every slot boundary from both sequences, RDV and timing information can be gathered and analyzed to construct statistical results. Like in the previous section, the asynchronous threshold is also defined to be a percentage of a slot, more specifically, a percentage of the maximum slot.

Unlike in most works, the goal of the performance analysis is to construct a complete statistical analysis of the protocol, both for TTR and RDV metrics. Therefore, every asynchronous analysis of two sequences is performed for every possible offset. Since the slots have a different size, an asynchronous offset needs to be considered, which is defined according to Equation 9.4, where t is the interval in number of microslots for every offset. The interval determines the number of iterations and the step size of the offset.

$$off_i = i \times t \qquad with \quad i = 0 \ldots \left\lceil \frac{cycle\ size}{t} \right\rceil - 1 \tag{9.4}$$

9.3.1 Asynchronous Rendezvous Extension

The benefits of an asynchronous system became clear in the previous section. The asynchronous behavior is a natural state thanks to entropy, the universal tendency towards disorder. However, this does not exclude the possibility of having two sequences that happen to be slot synchronized. The Asynchronous Rendezvous

Extension (ARE) is designed to instigate the asynchronism, even when SUs are slot synchronized. It is an extension to Rendezvous protocols that comply to some requirements, such that the slot size depends on the priority of the allocated channel. Therefore, even in a slot synchronized case, the sequences would operate in an asynchronous manner, except when the two sequences are completely frame synchronized. The opportunity for such occurrence is very small, depending on the length of the sequence, since it requires the two sequences to follow the exact same hopping sequence of equal length, while their periods start at the same instant.

The type of RDV protocols that are recommended for the Asynchronous Rendezvous Extension needs to meet some requirements. Since it is favorable that SUs rendezvous with each other as soon as possible, i.e., the Time-To-Rendezvous should be small and bounded, a guaranteed rendezvous on multiple channels is recommended. In such manner, the TTR decreases, since more RDV opportunities present themselves, while not relying on a single point of failure in case a PU should appear. Moreover, the activation of ARE results in a selective preference for the best channel, according to the channel order in the available channel list. Therefore, it is discouraged to use a RDV protocol that already provides preference to the best channel, since using both protocols could result in a deterioration of the performance.

The methodology of ARE is straightforward, the channel hopping sequence is constructed according to the RDV protocol it extends. The slot sizes, expressed in microslots, are determined by means of a channel ranking list and Equation 9.5, where *ch_priority* ranges from 0 to $N-1$, with 0 being the highest priority and N being the number of channels. The slot size variation (*ssv*) is a parameter that allows to adjust the amount of influence the extension has on the channel priority. When the *ssv* parameter is reduced to a value equal to 0, the protocol extension is disabled and the original RDV protocol behavior is enforced.

$$slot\_size = 100 - ssv * ch\_priority \tag{9.5}$$

This trivial equation is designed such that it allows for an approximately uniform decrease in slot size respecting a particular channel priority, while maintaining its flexibility by adjusting the asynchronous influence thanks to the *ssv* parameter. As a result of this extension, certain channels are assigned longer slots than others, where the largest slot always has a size equal to 100 microslots, i.e., 100% of the maximum slot size.

The influence of *ssv*, and therefore also ARE, is depicted in Figure 9.8, where one of the analyzed protocols (A-MOCH [245]) is analyzed in terms of the channel usage, that is, the RDV opportunity per cycle expressed in microslots. The *ssv* ranges from 0, i.e., the original protocol behavior, and 10, while the number of channels is equal to eight. Figure 9.8 clearly shows the intended purpose of the extension, that is, allowing more RDV time on the most significant channel, which is in this case "ch1", and gradually less RDV time on the worse channels, "ch2-8".

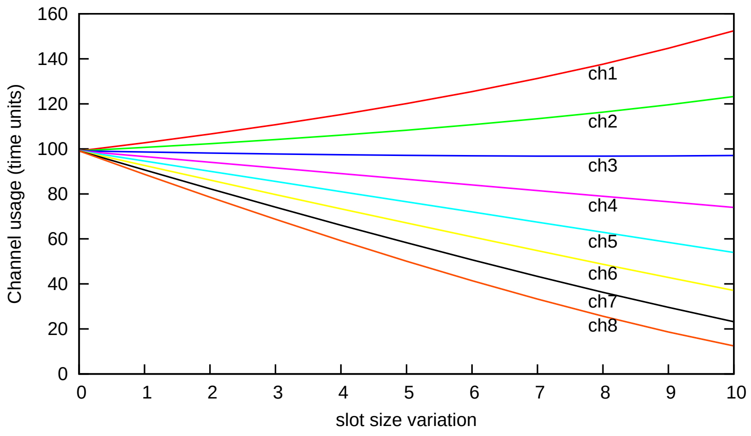


Figure 9.8: *Channel usage of a RDV protocol with ARE*

9.3.2 Performance evaluation

Since the asynchronous rendezvous extension (ARE) is designed for a rendezvous protocols that comply to specific requirements, the extension is evaluated in combination with three carefully selected RDV protocols. The specific protocols are MtQS-DSrdv [275], AMOCH [245], and DSMMAC [274] that assure a rendezvous at each channel in a cycle and do not support channel priorities. The latter is already induced by ARE, which can support a channel ranking in any rendezvous protocol. The performance is measured by the statistical analysis that is applied to the collected metrics, such as TTR, normalized number of rendezvous, and efficiency.

Like in the previous section, the case where a RDV occurs with two neighboring slots that happen to be allocated the same channel needs to be considered in the asynchronous case. Therefore the RDV opportunity, is defined as the number of rendezvous on sequentially distinct channels. Since the number of RDV opportunities per cycle is relative to the cycle size, which is often different for each protocol, it becomes an unsuitable metric to compare different protocols. Therefore, the normalized RDV, which is calculated according to Equation 9.6, is introduced in order to compare the protocols.

$$Normalized\ RDV = \frac{RDV}{cycle\_time} * 100 \tag{9.6}$$

Note that RDV in the equation stands for RDV opportunity and the *cycle_time* is the cycle duration expressed in microslots. In a channel hopping sequence where slots are equally sized, this metric would denote the number of RDV opportunities per slot. Later in this section, the normalized RDV performance is referred to when discussing the performance of the number of RDVs while comparing the studied protocols.

The most common discussed metrics for RDV protocols in CRNs are Time-To-Rendezvous (TTR), the time interval between two consecutive RDVs, which should be bounded and small, and the number of RDVs, which should be large. The performance discussion of this section includes a third metric, that is, the efficiency. It is defined according to Equation 9.7 and expresses the percentage of a cycle that is available for the sum of all RDVs (t_{sumRDV}).

$$Efficiency = \frac{t_{sumRDV}}{cycle\_time} \tag{9.7}$$

It should be noted that the performance analysis of ARE has been restricted to a single number of channels, that is, eight channels, since the construction of some of the selected protocol is a complex matter and proves to be a time-consuming task. The analyzed results for a series of number of channels of the protocols that allow for a less complex channel hopping sequence construction show that the behavior of the extension remains similar when compared with eight channels. Those results are omitted since the next section discusses the performance of the enhanced version of ARE in detail for a wide range of both protocols and number of channels.

Since ARE induces not only asynchronism, but also channel ranking, the performance is analyzed for two extreme channel ranking cases. The best case, later on referred to as 'same channel order', is where SUs observe similar channel qualities, resulting in a equally ordered channel lists. The worst case, also referred to as 'reverse channel order', is when one SU has a channel list in ascending order, while the other SU has a list in descending order.

All protocols are analyzed according to the discussed metrics by means of a custom simulation program, implemented in the C programming language. All simulations have been performed with the maximum asynchronous resolution, that is, for every possible offset lower than the cycle size, in steps of one microslot ($t = 1$, the finest granularity). The performance evaluation is executed for the tuned slot size variation parameter ranging from 0 to 10, and asynchronous thresholds ranging from 0 to 50. The *ssv* parameter equal to 0 signifies that the regular original protocol behavior is enforced. An asynchronous threshold (A_t) of 0 means that no matter the length of the RDV expressed in microslots, it is regarded as a valid RDV (i.e. $overlap > 0$). The synchronous analysis of an original protocol is equivalent to the analysis of the protocol with the *ssv* parameter equal to zero and the asynchronous threshold equal to 100.

The following subsections discuss the results from different angles. The first subsection presents the resulting TTR values in function of the slot size variation, while employing a static asynchronous threshold equal to 30 microslots. The following subsection discusses the results of TTR, efficiency and RDV in function of both the slot size variation and the asynchronous threshold, while the last subsection shows all three metrics for all three protocols for the lowest and largest slot size variation in function of the asynchronous threshold.

TTR versus slot size variation

This subsection examines the influence of the slot size variation on the TTR of all three protocols with an asynchronous threshold equal to 30 microslots. Figure 9.9 depicts the statistical analysis of all three protocols in the same channel order according to the resulting TTR values: minimum, maximum, average and 95% confidence interval. The red boxes indicate the 95% confidence interval of all measured values for all offsets. The red line within those boxes indicates the average TTR value and the blue lines extend toward the minimum and maximum values. The average values are highlighted in the zoomed figure at the top of the figure, in order to clearly depict the behavior of the average TTR of all three protocols.

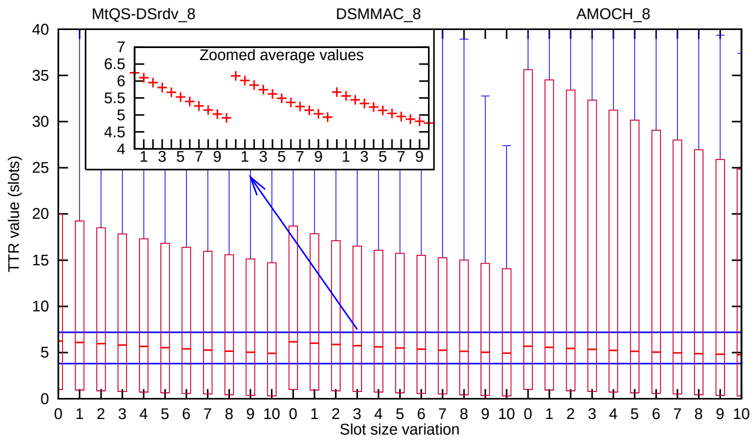


Figure 9.9: *TTR vs ssv in the same channel order*

It can be noted that both the maximum values of the confidence interval and the average TTR decreases for all three protocols with an increasing *ssv*, i.e., the more ARE is activated, the larger the improvement. The maximum values show the same decrease, but are cropped for the clarity of the more significant values. Although all protocols experience an improvement, the slope of the improvement is clearly different for each of the protocols. The difference between protocols will be discussed in more detail later.

The reverse channel order, which is the worst case situation, is depicted in Figure 9.10, where also the minimum, average, maximum and 95% confidence interval are shown. The DSMMAC protocol has been omitted from the reverse channel order analysis, since the protocol has an asymmetric number of slots per channel, i.e., not all slots are assigned an equal number of slots. By changing the channel order and applying ARE to the protocol, a different cycle length could be obtained. Such type of case requires analyzing the channel hopping sequences for a duration of the lowest common multiple of both sequences, instead of a single cycle. When analyzing the performance for every possible asynchronous offset, which is one microslot, the process would be time-consuming while no additional valuable results are obtained.

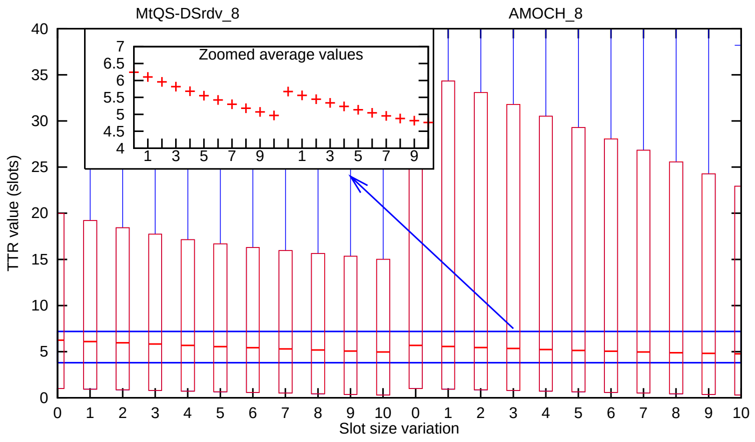


Figure 9.10: *TTR vs ssv in the reverse channel order*

As the figure indicates, even in the worst case scenario the TTR can be improved compared to the original protocol behavior (slot size variation equal to zero). The improvement of the TTR is a fraction smaller compared with the case where both nodes have channels in the same order, which is to be expected. The results of both cases, the same channel order and reverse channel order, are remarkably similar, indicating the effectiveness of ARE at an asynchronous threshold of 30 microslots.

Efficiency, RDV and TTR versus slot size variation and asynchronous threshold

Although the results of the previous subsection were positive, the *ssv* was only compared with a single asynchronous threshold. To obtain an overview of the intricacies of ARE and its behavior, simulations have been performed for all possible asynchronous thresholds between 0 and 50 microslots. The measured metrics are the RDV, TTR and efficiency. The simulations have been performed for all three protocols, of which only the results for MtQS-DSrdv are shown since the behavior of all three protocols is similar. An analysis based upon extremities of all three protocols can be found in Section 9.3.2.

Figure 9.11 depicts the influence of the slot size variation (*ssv*) and the asynchronous threshold (A_t) on the number of RDVs, i.e., the normalized RDV opportunities, of the enhanced RDV protocol for eight channels in the same channel order. The original protocol, that is, without an active ARE extension, is represented by a slot size variation of 0. From the results it is clear that there is an improvement in the number of RDVs, provided the asynchronous threshold is not too high.

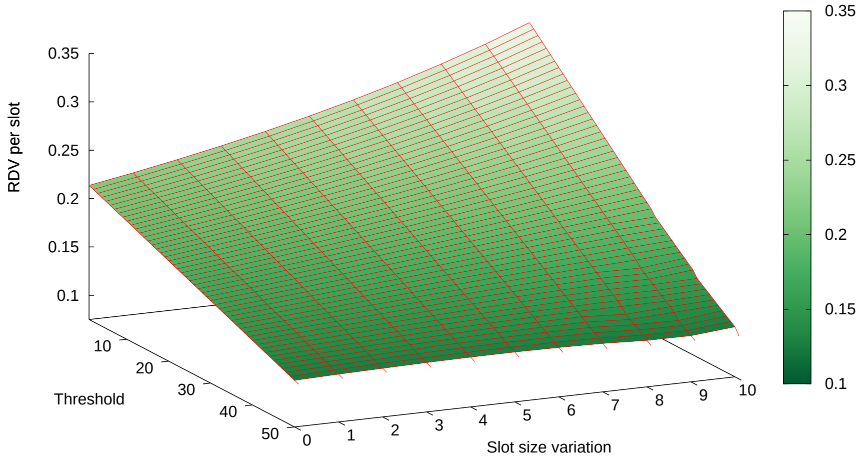


Figure 9.11: *RDV of MtQS-DSrdv vs ssv and A_t in the same channel order*

The operation of ARE is such that the highest priority channel maintains its slot size, that is, the maximum slot size, while all other channels obtain smaller slot sizes. This ensures that the total cycle time decreases in comparison to the original protocol. If the amount of time available for a rendezvous remains fixed, this results in an increase in the performance of the normalized RDV opportunity. The numerical results show that the amount of time available for RDVs is not the same, it is smaller, however, the decrease in cycle time is larger than the decrease in RDV time and results therefore in a performance improvement.

Note that the improvement decreases with a decreasing *ssv* and an increasing threshold. The former is related to the amount of adoption of ARE, while the latter is trivial; a larger threshold reduces the number of overlaps to comply to the threshold requirement. At a certain point, with a rising threshold, the application of ARE results even in a degradation of the RDV performance. This tipping point of the asynchronous threshold is situated around 50 microslots for MtQS-DSrdv for the RDV parameter. Note that the tipping point for each indivual protocol is studied into more detail in the following section.

By considering the performance improvement on the RDV opportunities, the Tim-To-Rendezvous (TTR) is also expected to improve by employing ARE. The influence of the *ssv* and the asynchronous threshold (A_t) on the TTR of MtQS-DSrdv in combination with ARE is depicted in Figure 9.12. As expected, the performance of the TTR increases, i.e., the TTR decreases, by applying ARE to the RDV protocol. The asynchronous RDV Extension (ARE) manipulates the slot sizes of low priority channels. Since the slots become smaller, a RDV is achieved faster when measured in absolute time and therefore the performance of the TTR increases.

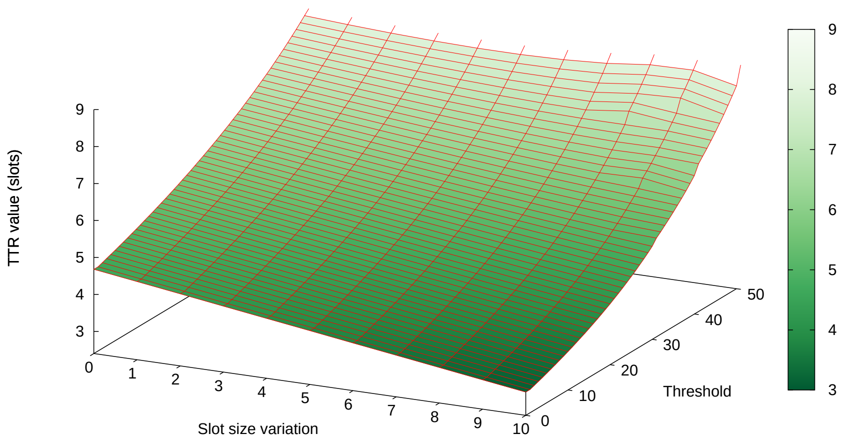


Figure 9.12: *TTR of MtQS-DSrdv vs ssv and A_t in the same channel order*

Like with the RDV performance, the magnitude of the performance improvement of the TTR is influenced by the *ssv* and asynchronous threshold. The amount of improvement is maximized when the ARE is maximally applied, while requiring a small asynchronous threshold. The improvement decreases with an increasing asynchronous threshold and after a tipping point, which is specified around 50 microslots for the TTR parameter of MtQS-DSrdv, the performance gain transforms into a performance loss when compared to the original protocol.

The third parameter that has been measured is the efficiency of the combination of protocols, defined as the percentage of time of the cycle duration that is available for the sum of all RDVs. It is necessary to verify this metric, since the protocol extension reduces the slot size of low priority channels and therefore changes the cycle length. It gives an idea whether the relative time available for RDVs exceeds, equals or is lower than the relative time available for RDVs of the original protocol. The efficiency of the combination of protocols is depicted in Figure 9.13, in function of the dependence on the slot size variation (*ssv*) and the asynchronous threshold (A_t). Since the original protocol is represented by the slot size variation equal to 0, it is clear that although ARE reduces the slot sizes, the relative amount of time available for RDVs is larger thanks to the application of the protocol. The protocol extension allows for a higher channel usage, when a rendezvous takes place, for high priority channels. Those high priority channels have larger slot sizes, therefore, the proportion of total overlap versus the total unused time becomes larger, leading to an increase in the efficiency.

Like with the RDV and TTR performance, such improvement is only valid for a

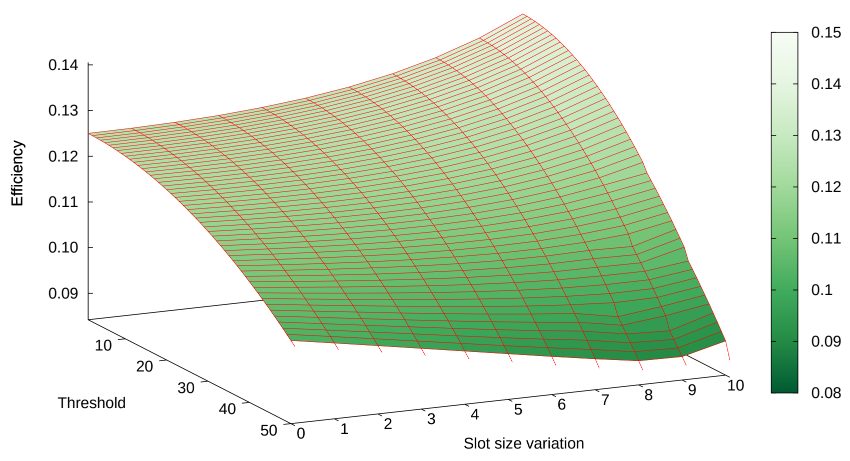


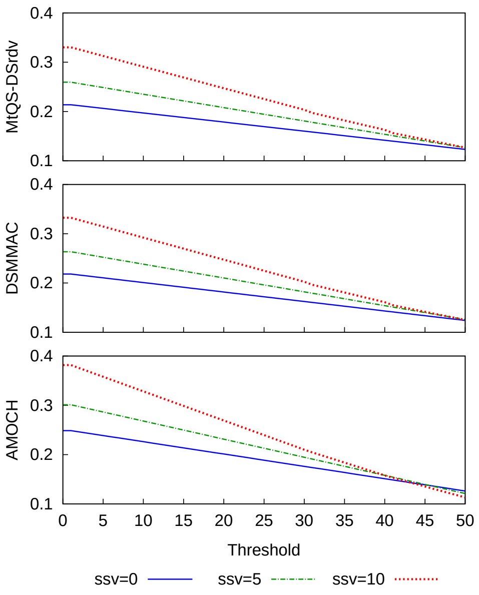
Figure 9.13: *Efficiency of MtQS-DSrdv vs ssv and A_t in the same channel order*

bounded asynchronous threshold. The tipping point where the improvement in performance becomes a degradation for the efficiency in combination with MtQS-DSrdv is situated around 35 microslots. Note that the tipping point of the efficiency is lower than the tipping point of both the TTR and RDV. An analysis regarding these tipping points is discussed in the following section for each inidivual protocol. Since the slots of the lower priority channels become smaller, the RDVs at those channels are reduced in time, thereby reducing the total time available for the sum the RDVs. Such behavior makes the efficiency metric more sensitive to the asynchronous threshold.

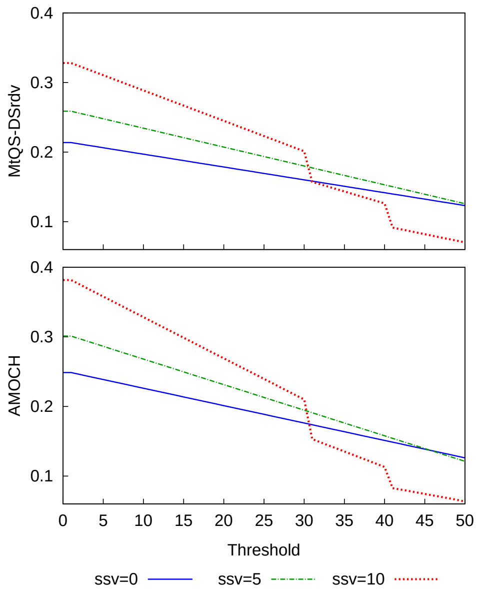
The observation of the performance of the three discussed metrics leads to the conclusion that the performance of the original protocol improves concerning all three metrics when applying the Asynchronous RDV Extension (ARE). The higher the slot size variation, i.e., the more ARE is applied, the higher the gain in performance. However, the asynchronous threshold needs to be bounded, since the performance deteriorates when the asynchronous threshold exceeds a certain tipping point, which is different for each of the metrics and each of the protocols. Since this parameter determines whether ARE signifies an improvement or deterioration, the next subsection studies in detail the tipping point for each of the protocols and each of the metrics.

Efficiency, RDV and TTR versus asynchronous threshold

Because of the significance of the asynchronous threshold regarding the performance, this subsection is dedicated to the comparison of the original protocol and the enhanced protocol with both an intermediate *ssv* and the maximum *ssv* parameter, that is, an *ssv* equal to 5 and 10. The depicted analysis will provide a deeper understanding of the asynchronous threshold tipping points of the different protocols and metrics. Provided the chosen threshold maintains beneath this tipping point, the Asynchronous Rendezvous Extension (ARE) results in an improved performance. The analysis is performed for both the same channel and reverse channel order.



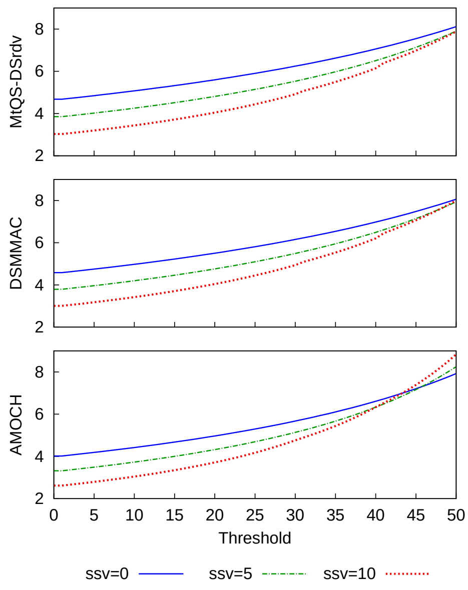
(a) *RDV same channel order*



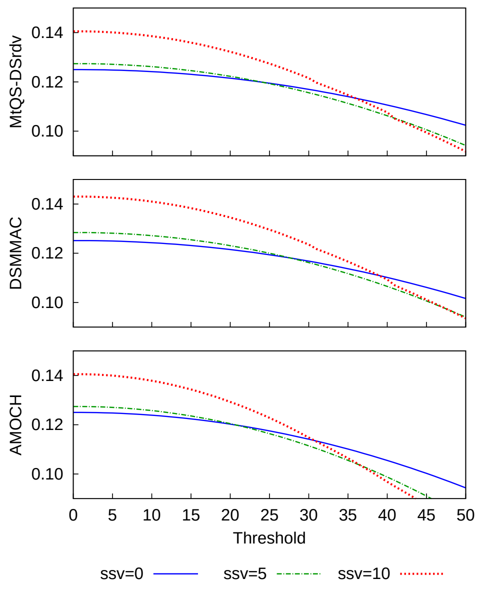
(b) *RDV reverse channel order*

Figure 9.14: *Asynchronous threshold tipping point for RDV*

The normalized RDV opportunity analysis is depicted in Figure 9.14 for both the same channel order (a) as the reverse channel order (b). The original behavior of the protocol is represented by the solid blue line, while the dashed green line represents the protocol with an enhancement of *ssv* equal to 5. The maximum enhancement, *ssv* equal to 10, is represented by the dotted red line. From the same channel ordering, figure on the left (a), it is clear that the tipping point for both MtQS-DSrdv and DSMMAC for RDV is positioned around 50 microslots. While the performance of A-MOCH is in general higher, its tipping point comes earlier, therefore allowing only for lower asynchronous thresholds. When comparing the same channel order with the reverse channel order, right figure(b), two characteristics stand out. The tipping point of the enhanced protocol with an *ssv* equal to 10 is lower compared to the tipping point in the same channel order. However, while the asynchronous threshold results show already a degradation in performance for the maximum enhanced protocol, it still ensures a performance improvement for the enhanced protocol with an *ssv* equal to 5.



(a) *TTR same channel order*



(b) *Efficiency same channel order*

Figure 9.15: *Asynchronous threshold tipping point for TTR and efficiency*

Figure 9.15 shows the performance analysis for both the TTR and efficiency in the same channel order. The tipping point of the TTR metric is similar for all protocols to the tipping point on the RDV metric, that is around 50 microslots for both MtQS-DSrdv and DSMMAC, and around 42 microslots for A-MOCH. The efficiency diagram (right side of the figure (b)), shows that the tipping point for all protocols is lower than their respective tipping point for the RDV or TTR. Therefore, it is possible to improve both the TTR and the RDV, while the efficiency is already degrading. As a consequence, there is relatively fewer total time available to achieve a RDV with the enhanced protocol when compared to the original protocol. However, the available time to rendezvous is more evenly distributed over the complete cycle, thereby increasing the number of RDVs and decreasing the TTR.

In the most extreme case, i.e., in a reverse channel order, the efficiency of all three protocols is worse for the enhanced protocol, when compared to the original protocol. From the start, the efficiency is worse, although both the RDV (depicted in Figure 9.14(b)) and TTR show a clear improvement until a certain asynchronous threshold. Even though the efficiency of the enhanced protocol is lower than the efficiency of the original protocol, improvement regarding RDV and TTR are achievable. Moreover, for each of the considered RDV opportunities, sufficient time is available to accomplish a RDV.

From the previous discussion, it can be concluded that the Asynchronous Rendezvous Extension (ARE) induces asynchronism to protocols with an equal channel opportunity per cycle by manipulating the slot size according to the channel ranking. The higher the priority of the channel, the larger the slot size. The protocol extension allows for a significant improvement in both RDV and TTR for all three considered protocols. While the efficiency of the protocols is worse in a reverse channel order, which is the worst case environment, both the RDV and TTR are able to achieve an improved performance, provided the asynchronous threshold is bounded. The more extensively ARE is used (the higher the *ssv* parameter), the larger the performance improvement. The higher the asynchronous threshold, i.e., the more time is required to have a RDV, the lower the performance improvement and can even lead to a degradation in performance when the asynchronous threshold becomes too high.

9.4 Enhanced Asynchronous Rendezvous Extensions

9.4.1 EAND hopping sequence

The Asynchronous Rendezvous Extension (ARE) was, besides the previously discussed three enhanced protocols, also evaluated in combination with a RDV protocol, named AND, specifically designed to work in cooperation with ARE in [251]. The RDV protocol (AND), designed by Dr. S.A. Romaszko, was a result of a thorough study of the performance of JS [267], SARCH [271] and Latin square (LS) based algorithms. The combination of the protocols and the extension showed that a performance could be achieved that exceeds the performance of most related work in its original state. However, AND is not optimal since it shows a certain preference towards an even number of channels. Therefore, in this section the improved version of AND, Enhanced Asynchronous Neighborhood Discovery (EAND), which is also designed by Dr. S.A. Romaszko, is concisely discussed where these shortcomings are largely removed. Note that this dissertation does not mention any contribution towards the development of EAND, since the contribution lies within the development of the Enhanced ARE protocol (discussed in the following section) and the performance analysis of the combination of Enhanced ARE and EAND. The combination of both protocols is discussed later in this chapter to determine the performance gain compared with related work.

EAND is guaranteed to have a RDV every cycle thanks to the combination of clockwise and counterclockwise channel sequences for every possible rotation. Moreover, an extra sequence, which is equal to half the number of channels, is introduced after every rotation section in order to provide a better coverage of the RDVs. Since the goal of this protocol is to be used in combination with the EARE, and thereby forming the E2AND protocol, it is important that the channels are uniformly distributed. At the end of the total sequence, a counterclockwise channel sequence is added to provide an even better channel distribution.

The pseudo-code of EAND can be found in Algorithm 1.

Algorithm 1 Enhanced Asynchronous ND

```
Require: List of available channels $C_0 \dots C_{N-1}$
Ensure: Channel sequence S
procedure EAND
    $S \leftarrow \varnothing$
    $ppow = prime\_power(N)$
    $lprim = lower\_prime(N)$
    $half\_num = \lfloor \frac{N}{2} \rfloor$
    for $i = 0$ to $N - 1$ do
        $r = i$
        for $n = 0$ to $N - 1$ do
            $idx = (r + 1) \bmod N$
            $S \leftarrow S \cup C_{idx}$
            if $iseven(N)$ then
                $r = (r + lprim) \bmod N$
            else
                $r = (r + ppow + 1) \bmod N$
            end if
        end for
        for $n = 0$ to $N - 1$ do
            $idx = (r + 1) \bmod N$
            $S \leftarrow S \cup C_{idx}$
            if $iseven(N)$ then
                $r = (r + ppow) \bmod N$
            else
                $r = (r + lprim + 1) \bmod N$
            end if
        end for
        for $n = 0$ to $half\_num - 1$ do
            if $iseven(N)$ then
                $idx = ((n \times i) - r) \bmod N$
            else
                $idx = ((n \times i) - r + 1) \bmod N$
            end if
            $S \leftarrow S \cup C_{idx}$
        end for
        $idx = (r \times i) \bmod N$
        $S \leftarrow S \cup C_{idx}$
    end for
    for $i = 0$ to $N - 1$ do
        $idx = (i - 1) \bmod N$
        $S \leftarrow S \cup C_{idx}$
    end for
end procedure
```

9.4.2 Enhanced Asynchronous Rendezvous Extensions

The Asynchronous Rendezvous Extension (ARE), which was extensively analyzed and discussed in Section 9.3.1, allows to introduce asynchronism where otherwise slot synchronized cases could occur. Since the benefit of exploiting the asynchronous character is already extensively discussed in this chapter, it is clear that such extension is able to provide a significant performance improvement. However, due to the trivial approach of manipulating the slot sizes, the extension is suboptimal. The distribution of the slots sizes is done according to Equation 9.8, where *ch_priority* ranges from 0 to $N-1$, where 0 stands for the highest priority, and N is the number of channels. The slot size variation (*ssv*) is a parameter that allows to tune the amount of influence the extension has on the channel priority. The highest ranking channel is therefore assigned a slot of 100 microslots, while the lower ranking channels are assigned a slot size depending on their level below the highest ranking channel and the *ssv* parameter.

$$slot\_size = 100 - ssv * ch\_priority \tag{9.8}$$

Note that the slot size depends on the number of levels below the highest ranking channel, i.e., in the extreme case of the lowest ranking channel, the slot size depends on the number of channels. However, the number of channels is not taken into account in Equation 9.8 to determine a lower bound. Therefore, the comparison of a protocol in combination with the extension for a series of number of channels is hard to accomplish. Either the *ssv* needs to be adjusted such that the configuration with the largest number of channels still has a positive slot size value for the lowest ranking channel, or the *ssv* parameter needs to be adjusted to the number of channels for each configuration. The former solution results in a suboptimal behavior for the protocol with the lower number of channels, which could achieve a better performance by increasing the *ssv* parameter. The latter results in an optimal setting for the complete series of number of channels, however proves hard to compare to each other due to the different *ssv* values.

In order to resolve this issue, three new rules to determine the slot size based upon the channel ranking are suggested in this section. All three methods take the number of channels into account and provide an absolute lower bound for the lowest ranking channel. While the highest ranking channels maintain a slot size of 100 microslots, all intermediary channels are assigned a slot size between the highest slot size and the lowest slot size, according to a certain distribution, relative to their channel ranking. The three proposed methods follow respectively a linear, exponential and logarithmic distribution.

The linear method (Equation 9.9), allows for a uniform decrease in slot size respecting a particular channel priority, that is, the maximum slot size difference ($100 - mss$) divided by the number of available channels minus one. The minimum slot size (*mss*) is a parameter that can be tuned to optimize the performance and represents the minimum slot size that is allowed for the lowest priority channel. This method is very similar to ARE, both distributions are linear, but very different because of its methodology of determining the slot size inter-spacing based upon the minimum slot size.

$$slot\_size = 100 - ch\_priority \times \frac{100 - mss}{N - 1} \tag{9.9}$$

The logarithmic method (Equation 9.10), provides longer slot size differences for a small number of high priority channels and smaller slot size differences for a large number of low priority channels, when compared to the linear method. This method ensures that only a few channels have a large slot size. As a consequence, the cycle size will be lower than the cycle size of the linear method, since there are more channels with a small slot assignment, while this is evenly distributed by the linear method. When assuming the number of RDVs remain the same, the reduction of slot size and cycle size also leads to a lower TTR. However, due to the fast decreasing slot size of the high ranking channels, the opportunity of effectually having a RDV could be reduced.

$$slot\_size = 100 - \frac{\ln(ch\_priority)}{\ln(N)} \times (100 - mss) \tag{9.10}$$

The last method, an exponential method (Equation 9.11), provides longer slot size differences for a large number of high priority channels and smaller slot size differences for a small number of low priority channels, compared to the linear method. Note that the factor of 1.15 has been experimentally determined such that it provides a graph where the difference between ssv parameters is inversely proportional to the difference between ssv parameters in the logarithmic method, i.e., when there is a small difference between two channels in the exponential method, there is a large difference in the logarithmic method. The method results in a larger number of channels obtaining a large slot size. This method ensures a larger cycle, compared to the cycle of the linear method. However, because of the larger number of channels with a large slot assigned to, the opportunity to have RDV on one of those high ranking channels is increased. By having more RDVs, preferable distributed over the complete cycle, the TTR could be reduced even though the slot and cycle sizes are increased.

$$slot\_size = 100 - \frac{1.15^{ch\_priority} - 1}{1.15^{N-1} - 1} \times (100 - mss) \tag{9.11}$$

An overview and the respective effect on the slot size decrease of all three methods is shown in Figure 9.16, where the slot size decrease is depicted up to 20 channels and

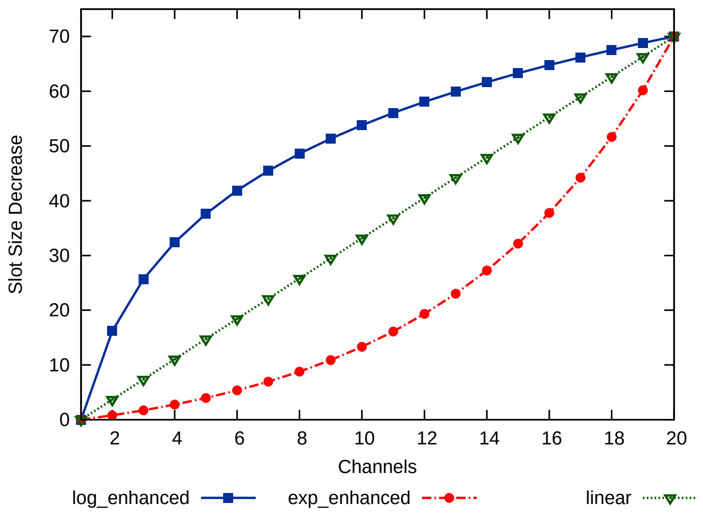


Figure 9.16: *EARE methods: slot size decrease for an mss value of 30*

an *mssv* value of 30. The resulting slot size is determined by subtracting this slot size decrease from the maximum slot size, that is, 100 microslots. The logarithmic method rapidly increases at first and stagnates later on, thereby leading to fewer channels with large slot sizes. On the other side, the exponential method results in a slow increase at first and a more rapid increase later on, resulting in more channels with large slot sizes. Finally, the linear method results in a linear decrease of the slot size.

9.4.3 Performance Evaluation

Like in the previous sections, the performance analysis makes use of microslots, that is, a subsection of a slot, where a single microslot is equal to 1/100 of a full slot. This allows to specify different slot sizes and to measure the number of microslots available for a RDV opportunity, that is, the number of rendezvous on sequentially distinct channels. The asynchronous threshold, which determines whether a RDV can be achieved during the overlap between two channel hopping sequences, is also expressed in number of microslots. The whole performance section assumes an asynchronous threshold of 30, unless otherwise specified. Since the different cycle sizes of all compared protocols make it hard to compare the RDVs, the concept of normalized RDV is employed. Normalized RDV is defined here as the number of RDV opportunities divided by the length of a cycle, expressed in microslots. All measurement results are subjected to a full statistical analysis for all possible map offsets with a resolution of one microslot.

The performance of the Enhanced Asynchronous Rendezvous Extension (EARE) is evaluated in combination with the EAND protocol, the combination of which is referred to as E2AND or eeand in the figures. Two types of evaluations are performed. In the following section, the three EARE methods are compared to each other and ARE in terms of normalized RDV, TTR and cycle size with different minimum slot sizes (*mss*). The outcome of this analysis will show the most suitable method to optimize the improvement of both RDV and TTR.

In Section 9.4.3 the best method from the previous analysis is compared to the performance of related work in terms of normalized RDV, TTR and cycle size. In order to evaluate the proposed solution in comparison with different ND approaches, rendezvous protocols are selected without a need of synchronization and distinctly outstanding among other related work in either having a better TTR, providing a guarantee on each channel in a single cycle, or considering the asynchronous character of CRNs. The selected protocols are already described in 8.4 and therefore will not be discussed here. The selected protocols to which the proposed solution is compared to are: JS [267][268], EJS [249], SARCH [271], DRDS [266], ETCH [265] and a variation of the protocol called mod_etch, such that all channels can be used, and A-MOCH [245]. Note that neither DSMMAC [274] nor MtQS-DSrdv [275] are included in the performance analysis due to their complex construction methodology.

Enhanced AREs performance

In this section, each of the EARE variants are evaluated in combination with the EAND Neighborhood Discovery (ND) protocol, as well as ARE in combination with EAND. Both protocol combinations are denominated as '*eeand*' in the figures. Each of the asynchronous extensions is named according to its distribution, the linear extension is called *linae*, the logarithmic extension *logae*, the exponential *expae* and ARE is just called *ae*. In order to discern the different protocol combinations, a concatenation is made of the ND protocol, the (E)ARE variant and the selected *ssv* or *mss* parameter.

All analyses have been performed for the number of channels ranging between 3 and 20. ARE is verified with an *ssv* of 5, which proves to have the best performance, while keeping within the boundaries dictated by the maximum number of channels. The three enhanced ARE variants are verified with an *mss* ranging from 5 to 30, signifying that the smallest slot for the lowest priority channel is at least 5 microslots when *mss* is equal to 5.

At first, the influence of the *mss* parameter on the performance regarding both normalized RDV and TTR is investigated for all three EARE variants in order to be able to make use of the best possible enhancement. Figure 9.17 depicts the RDV performance of the EAND protocol, enhanced with the *logae* with different values for the *mss*. It can be noticed that an *mss* value of 5 gives the best performance for all channels. The larger the parameter is, the lower the improvement becomes.

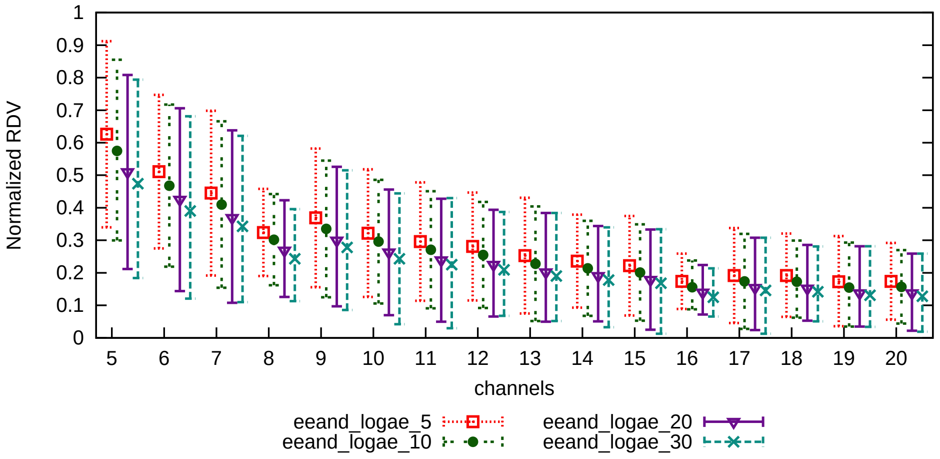


Figure 9.17: *The RDV of EAND with logae, with mss 5 till 30*

Since a lower *mss* parameter signifies a smaller minimum slot size, it is also related to a lower cycle size, since a lot of channels have a slot size in the neighborhood of the lowest slot size. When assuming a similar absolute number of RDVs, while the cycle is reduced, the normalized RDV is increased. The other EARE variants show similar results and are therefore omitted. Since the lower the *mss* parameter, the higher the performance, an *mss* parameter equal to 5 is selected to perform the rest of the performance comparisons. A minimum value of 5 is chosen, since an *mss* equal to 0 would eliminate the channel with the lowest ranking from the performance results.

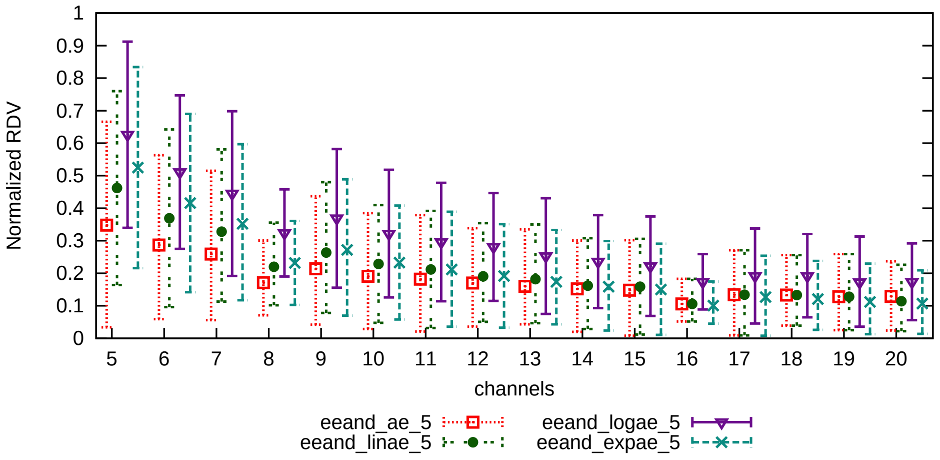


Figure 9.18: *The normalized RDV of EAND with (E)AREs*

The normalized RDV for all four asynchronous extensions is depicted in Figure 9.18. ARE is verified with *ssv* equal to 5, while the *mss* parameter is also chosen to be

5. Both the average value (point at the center) and the 95% confidence interval of the measured values are depicted in the figure. Since the *ssv* parameter of ARE is optimized for the largest number of channels, its performance is suboptimal. It exhibits the lowest performance of all, except for the largest number of channels, where it is better than the linear distribution. Interesting to note is the difference in performance between the linear and the exponential extension. The extension with the best performance of the RDV between both of them is continuously alternating. The extension with clearly the best performance regarding the normalized RDV is the logarithmic extension, when compared with all other extensions. Both its average number of RDVs and its maximum 95% confidence interval is higher than all other protocols.

As a result of decreasing the slot sizes according to the channel ranking, the total cycle size decreases as well. A comparison of the cycle sizes of all four extensions in combination with the EAND ND protocol is depicted in Figure 9.19. It can be noticed that the logarithmic extension provides the smallest cycle sizes, whereas ARE exhibits the largest cycle sizes, except for the largest number of channels, where the linear and exponential extensions have a larger cycle size. This is once more a result of the suboptimal behavior of ARE for all possible number of channels with an *ssv* equal to 5.

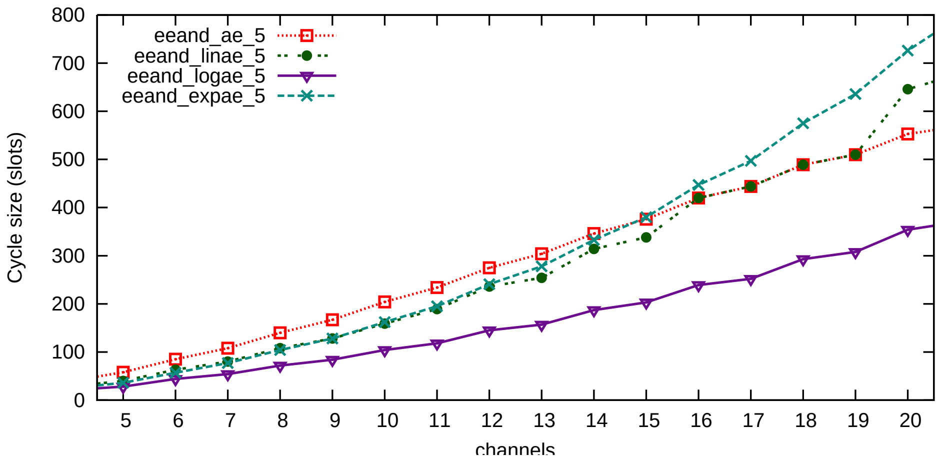


Figure 9.19: *The cycle size of EAND with (E)AREs*

Since the EAND protocol was designed such that all channels should have an equal amount of RDVs and the logarithmic extension allows only a few channels to have a large slot size and allows more channels to have smaller slot sizes where the relative difference is small, the cycle size is the smallest of all four extensions. Thanks to such reduction in cycle size and the asynchronous characteristics, the normalized RDV improves. Even when maintaining the time available for RDVs, while decreasing the cycle time, the resulting normalized RDV increases.

The Time-To-Rendezvous, depicted in Figure 9.20, shows that these improvements also count for the TTR, meaning that the TTR is the smallest with the logarithmic extension. Note that there is a reduction in improvement around 8 and 16 channels. This can be attributed to the ND protocol, EAND, which exhibits an imperfect distributed performance over all possible channels.

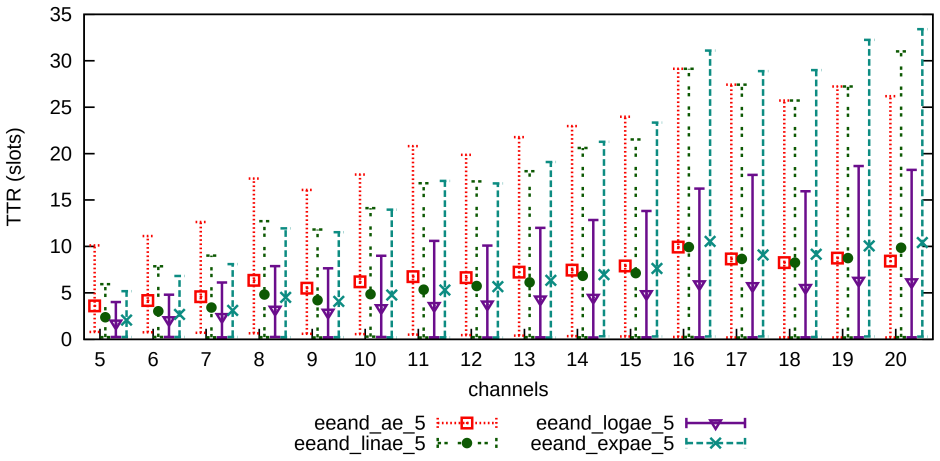


Figure 9.20: *The TTR of the EAND with (E)ARE*

From all analyzed material, it can be concluded that the EARE with a logarithmic distribution and an *mss* parameter equal to 5 gives the best performance. The fast decreasing slot size provides more RDV opportunity on the high ranking channels, while at the same time decreasing the Time-To-Rendezvous and the cycle size. As a result, the performance of both metrics is improved. While the larger number of big slots for high ranking channels with the exponential extension does provide a fair performance improvement in both RDV and TTR, its performance does not exceed the performance of the linear extension. Depending on the number of channels, the linear extension is better than the exponential extension or reverse.

Enhanced Asynchronous ND performance

This subsection is dedicated to the analysis of the EAND protocol in combination with the best asynchronous extension from the previous section, that is, the logarithmic extension. The combination of both protocols is called E2AND. The asynchronous extension is verified with the *mss* parameter equal to 5 and is referred to as eeand in the figures. All protocols are verified as in previous comparisons, that is, a statistical analysis is performed for every offset between two maps with a granularity of a single microslot. Since some of the protocols, such as Jump-Stay [267][268], ETCH [265], DRDS [266], SARCH [271], etc. include some randomness in their protocols, all performance analyses are repeated with the same parameters a 1000 times. The obtained results of each simulation run is considered in the statistical analysis. In order to be able to cope with the significant number of simulation runs, the simulations were performed on the w-iLab.t testbed in Zwijnaarde, Belgium.

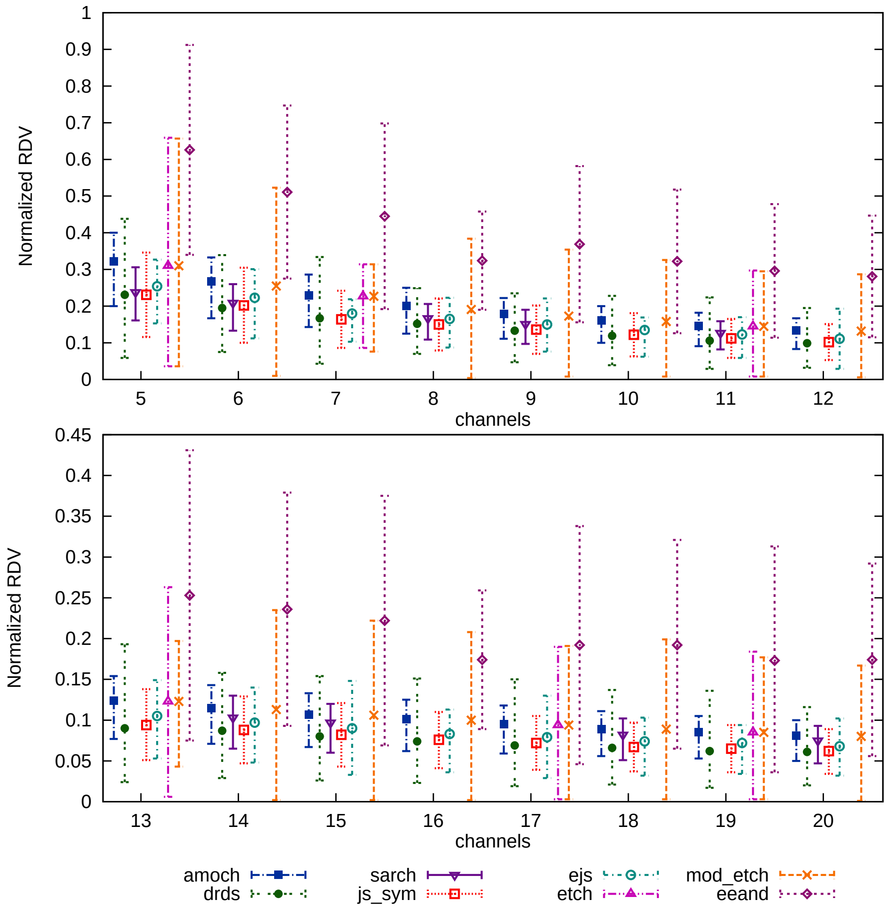


Figure 9.21: *RDV of E2AND with logae and mss 5 and related work*

The normalized RDV of all protocols can be seen in Figure 9.21, where it is explicit that the performance of E2AND significantly outperforms the other protocols. Even the average number of RDVs of the E2AND protocol almost always outperforms the upper bound of the 95% probability interval of the other protocols. Some of the evaluated protocols do not have values for certain channels because they pose restriction on the number of channels that can be used, such as ETCH that requires the number of channels is prime or SARCH that requires $2N + 1$ is prime, where N is the number of channels. The depicted results provide interesting insights , for example, while the protocol with the most absolute RDVs is SARCH, its normalized RDV is similar to the lowest performing protocols. The protocol with the lowest absolute RDVs is A-MOCH [245], while its normalized RDV is similar to the better performing protocols.

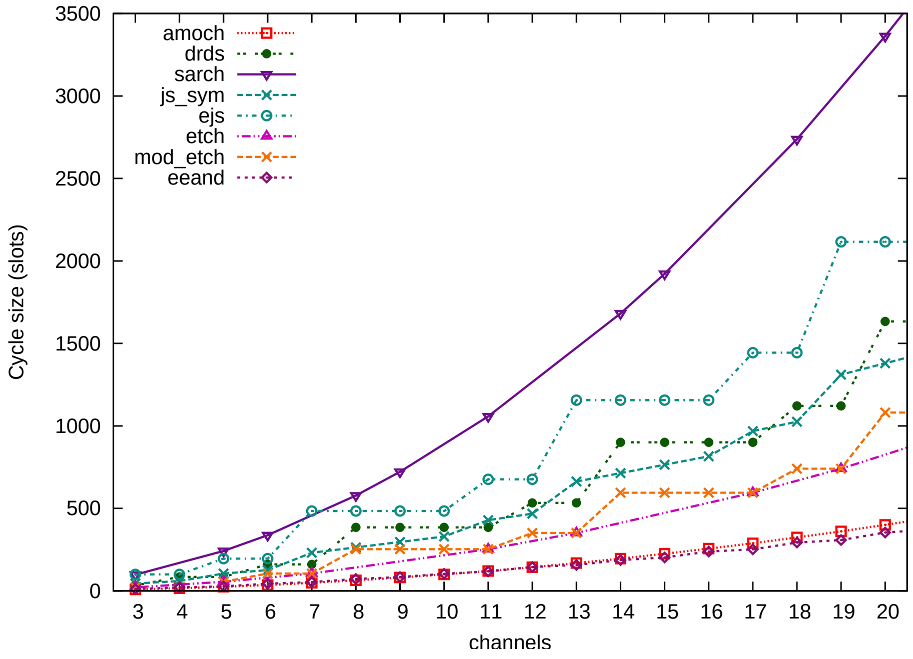


Figure 9.22: *Cycle size of E2AND with logae and mss 5 and related work*

As already mentioned before, the number of RDV is not a sufficient metric while comparing protocols with a different cycle size. Having a massive number of RDVs in a huge cycle does not necessarily mean that its performance in RDV is better, when compared to a protocol with a small cycle and a smaller number of RDVs per cycle. When following the hopping sequence of the small cycle during a number of cycles, such that the total simulation run equals the cycle length of the protocol with the largest cycle, the protocol with the smaller cycle could have achieved more RDVs than the protocol with the larger cycle. Such theory is also proven here, the cycle size of SARCH (Figure 9.22) is thus large in comparison to the cycle size of the other protocols, such that the normalized RDV of SARCH is situated around the same level as the other protocols. The cycle size of A-MOCH on the other hand is smaller, almost as small as the cycle size of E2AND, and therefore shows a better normalized RDV. Although the cycle size of A-MOCH and E2AND are nearly the same, the normalized RDV performance of E2AND is much higher than that of A-MOCH, indicating the performance improvement thanks to the induced asynchronism. Note that size does not always matter, as shown by EJS [249], where there is also a slight performance improvement regarding the normalized RDV of EJS when compared to JS, notwithstanding the larger cycle size of EJS.

The performance of the TTR (Figure 9.23) shows the same improvements. The TTR is far lower when using E2AND when compared to the other protocols. These results also prove the necessity of having a well designed ND protocol in combination with a low cycle size. Despite the low cycle size of A-MOCH, it exhibits a worse TTR

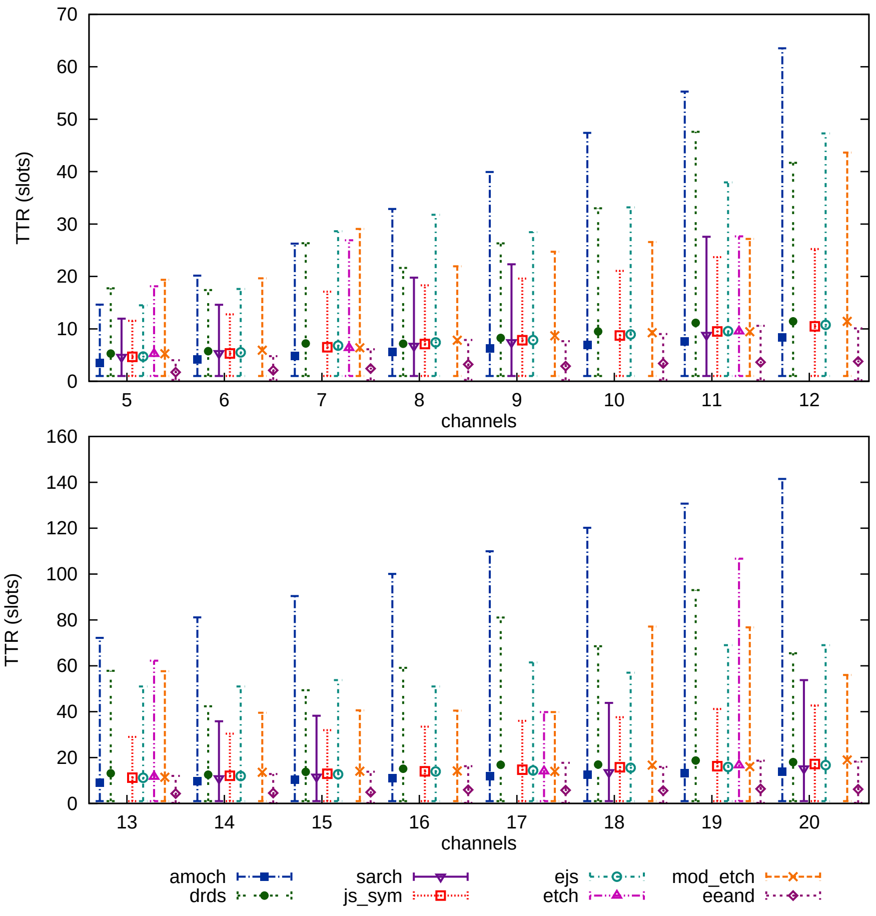


Figure 9.23: *TTR of E2AND with logae and mss 5 and related work*

than E2AND and especially its maximum 95% confidence interval is far higher than all protocols. The remainder of the protocol show an almost similar average TTR performance, while the maximum 95% confidence interval can deviate. Although it is not shown in the figure, interesting to note is that, while the Maximum TTR (MTTR) of JS, which is known for its very low MTTR, is superior to the MTTR of EAND, the MTTR of E2AND is similar or in some cases even better than the MTTR of JS.

The enhanced version of the protocol discussed in the previous section clearly portrays an improved performance. This section showed how to profit from added asynchronism in ND protocols. Thanks to induced asynchronism, the ranking of channels is introduced, allowing nodes to stay longer on better quality channels. The analysis of the behavior of different RDV protocols in terms of normalized RDV and TTR showed that the E2AND exhibits a significantly higher performance than any of the well known and discussed related work. As a result of the asynchronous extension was the cycle size significantly reduced, since less time was assigned to lower quality channels. This resulted in a significantly higher performance, since the number of RDVs was maintained.

9.5 Conclusion

Although only few works consider the asynchronous character of the nodes with a CRN, the discussed RDV protocols and extensions in this chapter show the possible improvements when it is taken into consideration. Note that the related work usually does not consider the time required to allow an effective message exchange. This might be justified if only synchronized slot arrangements are considered. However, when considering an asynchronous environment, the overlap between different slots needs to be taken into account. Three different approaches have been discussed and all three of them consider a certain asynchronous threshold, that is, the lower bound on the duration of the overlap which still guarantees sufficient time to make an effective message exchange.

The first approach discussed a method to analyze Rendezvous protocols in an asynchronous manner. The section does not consider protocols especially designed for asynchronous operation, it just indicates the performance improvement, especially in terms of TTR and RDV opportunity, when any protocol is not forced into a synchronous operation.

The second approach discusses the performance of certain RDV protocols which are enhanced by means of the proposed Asynchronous Rendezvous Extensions (ARE). Instead of defining a fixed slot size for all channels, each channel is assigned a dynamic slot size proportional to its relative priority. As such, channels with a higher priority are assigned a larger slot size. It can be noticed that the extension improves the performance of both RDV and TTR, provided that the asynchronous threshold is bounded.

A third protocol is a combination of a previously constructed protocol and an improvement of ARE. While ARE showed an improvement in both RDV and TTR, its application was limited by the number of channels. The same amount of improvement could not be applied to all possible channels. The improvement of ARE applies a logarithmic distribution in order to ensure its applicability for an entire range of channels. The protocol ensures an improved performance, since less time is granted to most lower priority channels, while only a few high priority channels are allocated a large slot size.

Note that none of the three works did assume the RDV to occur at the start of the hopping sequence, as some works do. A complete analysis is performed for every possible asynchronous offset in order to determine the actual maximum TTR and RDV for the protocols.

CHAPTER 10

Conclusions and Perspectives

This works focuses on the importance of timeliness and time in general in wireless networks. In order to demonstrate that its importance is not an isolated case, which is only applicable to a single type of network, is this work split up in three Parts, the first Part targeting Wireless Sensor Networks (WSNs), the second Part discussing Wireless Local Arera Networks (WLANs) and the third Part elaborating on the Time to Rendezvous (TTR) during the Neighborhood Discovery phase in Cognitive Radio Networks (CRNs). Each Part is introduced by an overview of the relevant state of the art and provides if necessary the necessary background to understand the context in which this work has been performed.

The Part about Wireless Sensor Networks is introduced by an overview of existing Wireless Sensor Network protocols, showing that a lot of diversity exists between the protocols. Both CSMA and TDMA based protocols are discussed, as well as some hybrid protocols that employ a combination of both. It is interesting to note that some protocols require the construction of a routing tree, while others prefer to use a two-hop neighborhood map in order to coordinate the access to the medium. The slot assignment of TDMA protocols can be either centralized or distributed. Some protocols try to minimize the time it takes to send a packet over multiple hops, by aligning the wake-up times of the next hop neighbors. And like that, there are a large number of protocols that use a somewhat different approach or have a different goal. Note that the list of described protocols is certainly not exhaustive, the discussed protocols provide an idea of possible MAC protocols, interesting and/or similar ideas as the concepts used to design the MAC protocol which is described in the third chapter of this Part on Wireless Sensor Networks, show an alternative way of achieving similar goals or take similar approaches.

The two issues that are being addressed in the Part about Wireless Sensor Networks are: efficient synchronization methods and the slot allocation in TDMA based heterogeneous sensor networks. The chapter on synchronization discusses the sections of the relevant hardware parts are are responsible for generating a stable clock signal to the microprocessor and why it is so important for the nodes within a sensor network to become synchronized. Numerous synchronization protocols are discussed and compared to each other, based on a theoretical model of the hardware. From those observations, it became clear that the existing protocols either require a non-negligible communication overhead, do not synchronize all nodes to each other, or rely on the exact timings of the hardware in order to perform a synchronization. A new synchronization method is proposed which eliminates any of those drawbacks by specifying a strict frame structure, and by specifying in the synchronization packet not the time at which it should be received, but the slot number within the frame. In such manner, the synchronization does not rely too much on the specifics of the hardware, but on the other hand, does not cause excessive communication overhead.

The overview of the available MAC protocols in WSNs show that heterogeneous networks are often not considered in the literature. For example, the slot assignments for scheduled channel access methods usually consider only a single slot per node, which might result in buffer overflows and therefore loss of data. To remedy the issue, the chapter which proposes a novel slot allocation mechanism for TDMA based heterogeneous networks, dives into the world of the scheduling algorithms, such as Weighted Fair Queuing (WFQ), Stop-and-Go queuing and packet-by-packet Generalized Processor Sharing protocol (PGPS), which were in the first place designed to provide a fair throughput in a router or switch, where multiple input sources would require access to the same output port. Since the proposed protocol makes use of bandwidth fragmentation, a protocol that constructs a cyclic binary sequence in a similar manner is discussed. A number of real-time MAC protocols are discussed, in order to be able to compare the real-time character of the proposed protocol to the state of the art regarding real-time MAC protocols.

The new MAC protocol that has been proposed, approximates the requested bandwidth and allocates the different fractions according to their frequency. The protocol makes use of fractional bandwidth allocations in order to approximate the requested bandwidth request. Assumed is that the slot size is fixed in order to prevent network synchronization issues and that the number of slots per frame is fixed and should be equal to a power of two. In order to distribute the bandwidth of a single frame, each bandwidth request is converted into binary Egyptian fractions, which is a series of unit fractions, where the denominator is a power of two. Each fraction represents a fraction of the available bandwidth per frame. By regarding each fraction as a frequency, a slot allocation can be made according to those fractions by assigning a first slot and allocating every 1/f slot for this bandwidth request. In such way that when the frequencies are applied, none of the allocations overlap, and the slot allocations interleave with each other, such that a node requiring a huge amount of bandwidth, is able to send its data on time, according to its bandwidth request, but at the same time, is not blocking the transmission of a node that requires a low bandwidth. On top of that, the resulting schedule can be encoded in a small number of bytes, which needs to be sent to the node only once, even if new nodes join or leave the network. Moreover, the protocol exhibits real-time properties, such as the deterministic latency.

The second Part of this dissertation focuses on the non-standardized operation of commodity hardware. Numerous related works have been provided which indicate the need to have a non-standardized operation. Thanks to the Linux kernel, such operation is made possible, of which the possible subsystems that might have an influence are discussed in more detail. An analysis of the possible timer sources lead to the conclusion that the hrtimer of the Linux kernel exhibits an very interesting performance, which is only exceeded by either the real-time core timer of the Xenomai real-time extension, or the timer of the eCos RTOS.

The enabling of a scheduled transmission in the Linux kernel and related hardware driver required a considerable number of modifications. A number of functionalities needed to be circumvented, others disabled, a new association mechanism was required, the radio reception needed to be disabled during the transmission period, etc. A careful analysis of the network operation in a standard environment showed several hardware operations which interfered with the timeliness of the transmission schedule. Therefore, amongst others, the beacon transmissions were integrated into the data schedule. The resulting transmission schedule was driven by the hrtimer, which triggered the next data transmission. Results showed that the modifications enabled a rather accurate scheduled transmission, where only a fraction of a percentage of the arrived packets showed a deviation of more than 10 µs compared to the expected transmission time.

The third Part discussed the performance of Rendezvous protocols in the context of a Cognitive Radio Network. The overview of existing methods showed the lack of consideration for the asynchronous operation. The first method proposed a low impact method to evaluate protocols in an asynchronous manner, showing that a considerable improvement could be achieved by allowing protocols to operate in an asynchronous manner. Therefore an asynchronous extension was proposed, which ensured that protocols would operate in an asynchronous manner by assigning larger slot to the higher priority protocols. The performance evaluation showed promising results, indicating the benefit of exploiting the asynchronous behavior. A further optimization of the protocol, in combination with a Rendezvous protocol specifically designed to cooperate with the extension, showed that the performance of the combination significantly outperformed any regular Rendezvous protocol.

Future work needs to be performed for all discussed protocols. The proposed TDMA based MAC protocol in Chapter 5 shows promising results when considering only a star topology. However, when a larger network is considered, the performance drops due to the extra slot allocations required to enable the parents to service their children. Moreover, although the latency can be controlled by limiting the accuracy of the approximation, the downside of this operation is that for a small number of bytes an entire slot is required. Possible improvements could be made by allowing a heterogeneous slot size throughout the network. In other words, when an entire slot is considered too large, half of a slot could be assigned. Since every transmission incurs some overhead regarding radio switching time and preamble transmission, such action will result in a lower efficiency, that is, less bandwidth will be available for data transmissions when compared to a full slot. However, when such method is employed only when absolutely necessary, it can result in a considerable improvement. On the other hand, since the protocol is optimal in a star formation, the network could be segmented in a number of clusters, making usage of one of the discussed methods to determine cluster heads, and the protocol would ensure a timely delivery of the data within each cluster.

The implementation of the scheduled transmission in the Linux kernel can also be improved in a significant manner. Currently, the reception mechanism of the radio is disabled during the transmission period in order to prevent any probe requests to capture the radio. A method similar as in one of the related works could be employed, where the receive path was directed to an empty antenna, thereby ensuring that the radio did not experience any interference from receptions. Moreover, the current implementation only involves the transmitter side. The receiver side should incorporate a synchronization mechanism based on the already established synchronization between the Access Point and its clients. The implementation showed also that when operating in a normal environment, where a lot of foreign traffic can be detected, a lot of packets are lost. In order to remedy this issue, the protocol could be adjusted to included a carrier sensing before initiating the transmission, as some of the WSN MAC protocols do. A more flexible slot assignment would be required in such case. Note that up till now, the entire implementation was done at the host side, that is, the Linux kernel. Recently, the ath9k_htc firmware source code has also been made public as open source, allowing to make an implementation of the scheduled transmission in the firmware, thereby circumventing any possible impact from kernel processes.

The Rendezvous protocol of the last section has been verified by means of simulations. However, implementing the protocol on hardware, such as USRP, would still provide considerable challenges regarding the actual detection of the SUs. Methods similar as Low Power Listening of WSNs could be employed for such application. However, the frequency of the transmissions and listen events should be carefully considered, since the simulation only considered that sufficient time is available to allow a communication. Initiating such communication only at the start and end of a slot would not even approach a performance similar to the one determined by the simulations.

During the time span of writing this work, 3GPP started on the 5G standard, which defines the design of ultra reliable and low latency communication (uRLLC). It is interesting to see whether the proposed algorithm in this work matches the concepts of this standard. In short, the key elements of uRLLC are described here and compared to our proposed protocol afterwards. The first key element of uRRLC is the usage of mini-slots, which are a lot smaller than usual slots. This allows for a flexible transmission duration. Moreover, is the UE (User Equipment) more frequently monitoring the DL (down-link) control to check to check the scheduling information for both the DL and UL (up-link) data transmissions. In order to minimize the waiting time to transmit a packet, the UE does not need to transmit a request first and then wait for a UL grant. Periodic UL resources for a UE allow the UE to transmit as soon as it has data available.

It is interesting to see how some ideas are comparable to the concepts, proposed in this dissertation. For example, instead of demanding that a node or UE requests for bandwidth every time, bandwidth is allocated for this node, such that it can send as soon as it has data available. In the protocol, described in this work, every node send its required bandwidth estimation and receives such bandwidth until its requirements change. Other than that, as part of the future work, it was proposed to make the slots smaller, such that we can achieve a lower latency at the cost of lower efficiency. In uRRLC, smaller slot sizes are also proposed, in order to allow more frequent scheduling of the resources. Thanks to the periodicity of the protocol in this work, there is already a frequent scheduling of the slots, not requiring any adjustment to that part of the protocol and therefore a more frequent monitoring of the schedule is not required.

References

[1] *The MADWIFI project.* madwifi-project.org.

[2] *Definition of timeliness according to Oxford Dictionaries.* https://en.oxforddictionaries.com/definition/timeliness.

[3] J. Drake, D. Najewicz, and W. Watts. *Energy Efficiency Comparisons of Wireless Communication Technology Options for Smart Grid Enabled Devices.* Technical report, General Electric Company, GE Appliances & Lighting, 2010.

[4] N. Kaabouch and W.-C. Hu. *Handbook of Research on Software-Defined and Cognitive Radio Technologies for Dynamic Spectrum Managements.* Advances in Wireless Technologies and Telecommunication (AWTT). IGI Global, 2015. ISBN 978-1-4666-6571-2.

[5] C. Stevenson, G. Chouinard, Z. Lei, W. Hu, S. Shellhammer, and W. Caldwell. *IEEE 802.22: The first cognitive radio wireless regional area network standard.* IEEE Communications Magazine, 47(1):130–138, February 2009.

[6] *Universal Software Radio Peripheral (USRP N210).* https://www.ettus.com/product/details/UN210-KIT. Accessed March 15, 2013.

[7] *Wireless Open-Access Research Platform (WARP).* http://warp.rice.edu/.

[8] A. S. Tanenbaum. *Computernetwerken.* Academic Service, Schoonhoven, 1999.

[9] C. Hsieh. *Course CIS370: Introduction to Computer Networks.* http://pluto.ksi.edu/~cyh/cis370/ebook/ch05b.htm. Accessed 24 June, 2015.

[10] NXP Laboratories UK. *ZigBee PRO Stack User guide*, 2014.

[11] R. Breyer and S. Riley. *Switched, Fast, and Gigabit Ethernet, Third edition.* Macmillan Network Architecture And Development Series. Macmillan Technical Publishing, 2002. ISBN 1-57870-073-6.

[12] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, March 2012.

[13] A. Burns and A. Wellings. *Real-Time Systems and Programming Languages, 3rd edition.* Pearson Education Limited, Essex CM20 2JE, England, 2001. ISBN 0-201-72988-1.

[14] H. Kopetz. *Real-Time Systems: design principles for distributed embedded applications.* Kluwer Academic Publishers Group, 3300 AH Dordrecht, The Netherlands, 1997.

[15] J. J. Labrosse. *Embedded Systems Building Blocks, 2nd edition.* CMP Books, Lawrence, Kansas 66046, USA, 2000. ISBN 0-87930-604-1.

[16] J. J. Labrosse. *MicroC/OS-II: The Real-Time Kernel, second edition.* CMP Books, Lawrence, Kansas 66046, USA, 2002.

[17] A. Bachir, M. Dohler, T. Watteyne, and K. K. Leung. *MAC Essentials for Wireless Sensor Networks.* IEEE Communications Surveys & Tutorials, 12(2):pp 222 – 248, 2010.

[18] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. *Wireless sensor networks: a survey.* Computer Networks, 38:pp 393 – 422, 2002.

[19] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury. *Wireless Multimedia Sensor Networks: A Survey.* IEEE Wireless Communications Magazine, 14:pp 32 – 39, 2007.

[20] X. Liu. *A Survey on Clustering Routing Protocols in Wireless Sensor Networks.* Sensors, 12(8):pp 11113 – 11153, 2012.

[21] P. Huang, L. Xiao, S. Soltani, M. W. Mutka, and N. Xi. *The Evolution of MAC Protocols in Wireless Sensor Networks: A Survey.* IEEE Communications Surveys & Tutorials, 15(1), 2012.

[22] W. Ye, J. Heidemann, and D. Estrin. *An Energy-Efficient MAC protocol for Wireless Sensor Networks.* In Proceedings of IEEE Computer and Communications Societies (INFOCOM), 2002.

[23] M. Adnan and E.-C. Park. *Improving Energy Efficiency in Idle Listening of IEEE 802.11 WLANs.* Mobile Information Systems, 2016:15 pages, 2016. Article ID 6520631.

[24] X. Zhang and K. G. Shin. *E-MiLi: Energy-Minimizing Idle Listening in Wireless Networks.* IEEE Transactions on Mobile Computing, 11(9):pp 1141 – 1454, September 2012.

[25] T. van Dam and K. Langendoen. *An Adaptive Energy-Efficient MAC protocol for Wireless Sensor Networks.* In Proceedings of International Conference on Embedded networked sensor systems (SenSys), Los Angeles, California USA, 2003.

[26] T.-H. Hsu, T.-H. Kim, C.-C. Chen, and J.-S. Wu. *A Dynamic Traffic-Aware Duty Cycle Adjustment MAC Protocol for Energy Conserving inWireless Sensor Networks.* International Journal of Distributed Sensor Networks, 2012:pp 10, 2012.

[27] J. Polastre, J. Hill, and D. Culler. *Versatile low power media access for wireless sensor networks.* In Proceedings of the 2nd international conference on Embedded networked sensor systems, SenSys, Baltimore, Maryland, 2004.

[28] W. Ye, F. Silva, and J. Heidemann. *Ultra-Low Duty Cycle MAC with Scheduled Channel Polling.* In International conference on Embedded Networked Sensor Systems, SenSys, 2006.

[29] B. Jang, J. B. Lim, and M. L. Sichitiu. *An Asynchronous Scheduled MAC Protocol for Wireless Sensor Networks.* Computer Networks, 12:pp 85 – 98, 2013.

[30] A. El-Hoiydi and J.-D. Decotignie. *WiseMAC: An Ultra Low Power MAC Protocol for the Downlink of InfrastructureWireless Sensor Networks.* In 9th International Symposium on Computers and Communications, ISCC, July 2004.

[31] S. Mishra and A. Nasipuri. *An Adaptive Low Power Reservation Based MAC Protocol for Wireless Sensor Networks.* In IEEE International Conference on Performance, Computing, and Communications, 2004.

[32] B. Latré, B. Braem, I. Moerman, C. Blondia, E. Reusens, W. Joseph, and P.Demeester. *A low–delay protcol for Multihop Wireless Body Area Networks.* In Mobile and Ubiquitous Systems: Networking & Services, 2007.

[33] D.-S. Yoo and S. S. Choi. *Medium Access Control with Dynamic Frame Length in Wireless Sensor Networks.* Journal of Information Processing Systems, 6(4):pp 501 – 510, 2010.

[34] A. Ephremides and T. V. Truong. *Scheduling Broadcasts in Multihop Radio Networks.* IEEE TRANSACTIONS ON COMMUNICATIONS, 38(4):pp 456 – 460, 1990.

[35] V. Rajendran, K. Obraczka, and J. Garcia-Luna-Aceves. *Energy-efficient collision-free medium access control for wireless sensor networks.* In Proceedings of the 1st international conference on Embedded networked sensor systems, 2003.

[36] V. Rajendran, J. J. Garcia-Luna-Aceves, and K. Obraczka. *Energy-Efficient, Application-Aware Medium Access for Sensor Networks.* In IEEE MASS, 2005.

[37] S. Chatterjea, L. van Hoesel, and P. Havinga. *AI-LMAC: An Adaptive, Information-centric and Lightweight MAC Protocol for Wireless Sensor Networks.* In Proceedings of the Intelligent Sensors, Sensor Networks and Information Processing Conference, 2004.

[38] L. van Hoesel and P. Havinga. *Collision-free Time Slot Reuse in Multi-hop Wireless Sensor Networks.* In Proceedings of International Conference on Intelligent Sensor, Sensor Networks and Information Processing, 2005.

[39] R. A. Rashid, W. M. A. E. W. Embong, A. Zaharim, and N. Fisal. *Development of Energy Aware TDMA-Based MAC Protocol for Wireless Sensor Network System.* European Journal of Scientific Research, 30(4):pp 571 – 578, 2009.

[40] I. Rhee and J. Lee. *Distributed Scalable TDMA Scheduling Algorithm.* Technical report, Computer Science Department, North Carolina State University, Raleigh, NC, 2004.

[41] I. Rhee, A. Warrier, J. Min, and L. Xu. *DRAND: Distributed Randomized TDMA Scheduling for Wireless Ad Hoc Networks.* In Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing, Florence, Italy, May 2006.

[42] W. L. Tan, W. C. Lau, and O. Yue. *Performance analysis of an adaptive, energy-efficient MAC protocol for wireless sensor networks.* Journal of Parallel and Distributed Computing, 72(4):pp 504 – 514, 2012.

[43] I. Rhee, A. Warrier, M. Aia, and J. Min. *ZMAC: a hybrid MAC for wireless sensor networks.* IEEE/ACM Transactions on Networking, 16(3):pp 511 – 524, 2008.

[44] H. Kopetz. *Real-Time Systems, 2nd edition.* Springer, 2011.

[45] S. Ganeriwal, R. Kumar, and M. B. Srivastava. *Timing-sync Protocol for Sensor Networks.* In Proceedings of the 1st international conference on Embedded networked sensor systems, 2003.

[46] L. Sitanayah, C. J. Sreenan, and K. N. Brown. *ER-MAC: A Hybrid MAC Protocol for Emergency Response Wireless Sensor Networks.* In Fourth International Conference on Sensor Technologies and Applications (SENSORCOMM), Venice, Italy, July 2010.

[47] W.-Z. Song, F. Yuan, and R. LaHusen. *Time-Optimum Packet Scheduling for Many-to-One Routing in Wireless Sensor Networks.* In IEEE International Conference on Mobile Adhoc and Sensor Systems, MASS, 2006.

[48] P. J. M. Havinga and G. J. Smit. *E2MaC: an energy efficient MAC protocol for multimedia traffic.* Technical Report Moby Dick technical report 1998, Electrical Engineering, Mathematics and Computer Science, University of Twente, 1998.

[49] G. Chakraborty. *Genetic Algorithm to Solve Optimum TDMA Transmission Schedule in Broadcast Packet Radio Networks.* IEEE TRANSACTIONS ON COMMUNICATIONS, 52(5):pp 765 – 777, 2004.

[50] S. Coleri. *PEDAMACS: Power Efficient and Delay Aware Medium Access Protocol for Sensor Networks.* Master's thesis, UNIVERSITY OF CALIFORNIA, BERKELEY, 2002.

[51] S. Coleri, A. Puri, and P. Varaiya. *PEDAMACS: Power efficient and delay aware medium access protocol for sensor networks.* IEEE Transactions on Mobile Computing, 5:pp 920 – 930, 2006.

[52] S. Coleri and P. Varaiya. *TDMA Scheduling Algorithms for Wireless Sensor Networks.* ACM Journal Wireless Networks, 16(4):pp 985 – 997, May 2010.

[53] G.-S. Ahn, E. Miluzzo, A. T. Campbell, S. G. Hong, and F. Cuomo. *Funneling-MAC: A Localized, Sink-Oriented MAC For Boosting Fidelity in Sensor Networks.* In Proceedings of the 4th international conference on Embedded networked sensor systems, 2006.

[54] J. Elson, L. Girod, and D. Estrin. *Fine-grained network time synchronization using reference broadcasts.* In Proceedings of the 5th symposium on Operating systems design and implementation, 2002.

[55] M. Salajegheh, H. Soroush, and A. Kalis. *HYMAC: Hybrid TDMA/FDMA Medium Access Control Protocol for Wireless Sensor Networks.* In IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC, 2007.

[56] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. *Energy-Efficient Communication Protocol forWireless Microsensor Networks.* In Proceedings of the 33rd Hawaii International Conference on System Sciences, 2000.

[57] M. Ye, C. Li, G. Chen, and J. Wu. *An Energy Efficient Clustering Scheme in Wireless Sensor Networks.* Ad Hoc & Sensor Wireless Networks, 3(2-3):pp 99 – 119, 2007.

[58] S. Mukherjee and G. P. Biswas. *Design of Hybrid MAC Protocol for Wireless Sensor Network.* Recent Advances in Information Technology, Advances in Intelligent Systems and Computing, 266:pp 11 – 18, 2014.

[59] J. Lessmann. *GMAC: A position based energy efficient QoS TDMA MAC for Ad Hoc Networks.* In IEEE International Conference on Networks (ICON), 2007.

[60] C. Shanti and A. Sahoo. *DGRAM: A Delay Guaranteed Routing and MAC Protocol forWireless Sensor Networks.* IEEE Transactions on Mobile Computing, 9(10):pp 1407 – 1423, 2010.

[61] G. Lu, B. Krishnamachari, and C. S. Raghavendra. *An adaptive energy-efficient and low-latency MAC for data gathering in wireless sensor networks.* In 18th International Parallel and Distributed Processing Symposium, 2004. Proceedings, April 2004.

[62] S. Du, A. K. Saha, and D. B. Johnson. *RMAC: A Routing-Enhanced Duty-Cycle MAC Protocol for Wireless Sensor Networks.* In IEEE INFOCOM, 2007.

[63] K. Nguyen, Y. Ji, and S. Yamada. *Low Overhead MAC Protocol for Low Data Rate Wireless Sensor Networks.* International Journal of Distributed Sensor Networks, 2013:pp 1 – 10, 2013.

[64] A. Keshavarzian, H. Lee, and L. Venkatraman. *Wakeup Scheduling in Wireless Sensor Networks.* In Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing, 2006.

[65] N. Gajaweera and D. Dias. *FAMA/TDMA Hybrid MAC for Wireless Sensor Networks.* In 4th International Conference on Information and Automation for Sustainability, ICIAFS, 2008.

[66] V. Turau and C. Weyer. *Scheduling Transmission of Bulk Data in Sensor Networks using a Dynamic TDMA Protocol.* In Proceedings of International Workshop on Data Intensive Sensor Networks(DISN), 2007.

[67] R. Kuntz, A. Gallais, and T. Noel. *Medium Access Control Facing the Reality of WSN Deployments.* ACM SIGCOMM Computer Communication Review, 39(3):pp 22 – 27, 2009.

[68] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. *Wireless Sensor Networks for Habitat Monitoring.* In Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications, WSNA, 2002.

[69] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. *Lessons From a Sensor Network Expedition.* In Proceedings of the First European Workshop on Sensor Networks (EWSN), 2004.

[70] T. Naumowicz, R. Freeman, A. Heil, M. Calsyn, E. Hellmich, A. Brändle, T. Guilford, and J. Schiller. *Autonomous Monitoring of Vulnerable Habitats using a Wireless Sensor Network.* In Proceedings of the Workshop on Real-World Wireless Sensor Networks (REALWSN), 2008.

[71] G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees. *Deploying a Wireless Sensor Network on an Active Volcano.* IEEE Internet Computing, 10(2):pp 18 – 25, 2006.

[72] R. Mancini. *Op Amps for Everyone.* Elsevier B.V., 2001.

[73] *Tutorial on The Quartz Crystal Oscillator.* `http://www.electronics-tutorials.ws/oscillator/crystal.html`.

[74] *Improving the Accuracy of a Crystal Oscillator.* Application Note. Semtech Application Note AN1200.07.

[75] *32.768kHz crystal datasheet.* MH32768B.

[76] P. Spevak and P. Forstner. *MSP430 32-kHz Crystal Oscillators.* Application Report, April 2009. SLAA322B.

[77] R. Cerda. *Pierce-Gate Crystal Oscillator, an introduction.* mpdigest, pages pp 1 – 3, March 2008.

[78] R. Aparicio and A. Hajimiri. *Capacity Limits and Matching Properties of Integrated Capacitors.* IEEE Journal of Solid-State Circuits, 37:pp 384 – 393, 2002.

[79] A. S. Tanenbaum and M. V. Steen. *Distributed systems: principles and paradighms.* Pearson, 2002.

[80] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: concepts and design.* Pearson Education Limited, Essex CM20 2JE, England, 2001.

[81] F. Cristian. *Probabilistic clock synchronization.* Distributed Computing, 3:pp 146 – 158, 1989.

[82] R. Gusella and S. Zatti. *The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD.* IEEE Transactions Software Engineering, 15:pp 847 – 853, 1989.

[83] D. L. Mills. *Internet time synchronization: the network time protocol.* IEEE Transactions on Communications, 39(10):1482–1493, 1991.

[84] D. L. Mills. *Improved Algorithms for Synchronizing Computer Network Clocks.* IEEE Transactions Networks, 39(10):245–254, 1995.

[85] K. Arvind. *Probabilistic clock synchronization in distributed systems.* IEEE Transactions on Parallel and Distributed Systems, 5(5):474–487, 1994.

[86] L. Lamport. *Time, Clocks, and the Ordering of Events in a Distributed System.* Communications of the ACM Magazine, 21:558–565, 1978.

[87] J. Elson and K. Römer. *Wireless Sensor Networks: A New Regime for Time Synchronization.* In Proceedings of the first workshop on Hot Topics in Networks, 2002.

[88] H. Kopetz and W. Schwabl. *Global time in distributed real-time systems.* Technical report, Technische Universität Wien, 1989.

[89] S. PalChaudhuri, A. Saha, and D. B. Johnson. *Probabilistic clock synchronization service in sensor networks.* Technical Report TR 03418, Department of Computer Science, Rice University, 2003.

[90] Q. Li and D. Rus. *Global Clock Synchronization in Sensor Networks.* In INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, 2004.

[91] Q. Li and D. Rus. *Global Clock Synchronization in Sensor Networks.* IEEE Transactions on Computers, 55:214–226, 2006.

[92] K. Römer. *Time Synchronization in Ad Hoc Networks.* In Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing, MobiHoc, 2001.

[93] M. L. Sichitiu and C. Veerarittiphan. *Simple, accurate time synchronization for wireless sensor networks.* In IEEE Wireless Communications and Networking, WCNC, 2003.

[94] S. Yoon and M. L. Sichitiu. *Analysis and Performance Evaluation of a Time Synchronization Protocol for Wireless Sensor Networks.* In Proceedings of the International Conference on Telecommunication Systems, Modeling and Analysis, ICTSM, Dallas, TX, November 2005.

[95] S. Yoon, C. Veerarittiphan, and M. L. Sichitiu. *Tiny-Sync: Tight Time Synchronization for Wireless Sensor Networks.* ACM Transactions on Wireless Sensor Networks, 3:1–33, 2007.

[96] J. van Greunen and J. Rabaey. *Lightweight Time Synchronization for Sensor Networks.* In WSNA '03 Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications, 2003.

[97] K.-L. Noh, E. Serpedin, and K. Qaraqe. *A New Approach for Time Synchronization in Wireless Sensor Networks: Pairwise Broadcast Synchronization.* IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, 7:3318–3322, 2008.

[98] L.-M. He. *Time Synchronization Based on Spanning Tree for Wireless Sensor Networks.* In 4th International Conference on Wireless Communications, Networking and Mobile Computing, WiCOM, Dalian, October 2008.

[99] S. N. Gelyan, A. N. Eghbali, L. Roustapoor, S. A. Y. F. Abadi, and M. Dehghan. *SLTP: Scalable Lightweight Time Synchronization Protocol for Wireless Sensor Network.* In Third International Conference on Mobile Ad-Hoc and Sensor Networks, MSN, volume 4864, pages 536–547, Beijing, China, December 2007.

[100] J. Hill and D. Culler. *A Wireless Embedded Sensor Architecture for System-Level Optimization.* Technical report, U.C. Berkeley, 2001.

[101] S. Ping. *Delay Measurement Time Synchronization for Wireless Sensor Networks.* Technical Report IRB-TR-03-013, Intel Corporation, June 2003.

[102] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi. *The flooding time synchronization protocol.* In Conference On Embedded Networked Sensor Systems, Baltimore, MD, USA, 2004.

[103] C. Lenzen, P. Sommer, and R. Wattenhofer. *Optimal clock synchronization in networks.* In Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, SenSys, 2009.

[104] W. Su and I. F. Akyildiz. *Time-Diffusion Synchronization Protocol for Wireless Sensor Networks.* IEEE/ACM TRANSACTIONS ON NETWORKING, 13:384–397, 2005.

[105] L. Schenato and F. Fiorentin. *Average TimeSynch: A consensus-based protocol for clock synchronization in wireless sensor networks.* Elsevier Journal Automatica, 47:pp 1878 – 1886, 2011.

[106] J. He, P. Cheng, L. Shi, and J. Chen. *Time Synchronization in WSNs: A Maximum Value Based Consensus Approach.* In IEEE Conference on Decision and Control and European Control Conference, CDC-ECC, 2011.

[107] J. He, P. Cheng, L. Shi, and J. Chen. *Study of consensus-based time synchronization in wireless sensor networks.* Elsevier Journal ISA Transactions, 53:pp 347 – 357, 2014.

[108] I. F. Akyildiz, D. Pompili, and T. Melodia. *Underwater acoustic sensor networks: research challenges.* Ad Hoc Networks, 3(3):pp 257 – 279, May 2005.

[109] M. A. Habib, J. Uddin, and M. M. Islam. *Safety Aspects of Enhanced Underwater Acoustic Sensor Networks.* International Journal of Emerging Technology and Advanced Engineering, 2(8):pp 385 – 390, August 2012.

[110] R. Manjula and S. M. Sunilkumar. *Issues in Underwater Acoustic Sensor Networks.* International Journal of Computer and Electrical Engineering, 3(1):pp 1793 – 8163, February 2011.

[111] A. A. Syed and J. Heidemann. *Time synchronization for high latency acoustic networks.* In Conference on Computer Communications, INFOCOM, 2006.

[112] J. Heidemann, W. Ye, J. Wills, A. Syed, and Y. Li. *Research challenges and applications for underwater sensor networking.* In IEEE Wireless Communications and Networking Conference, WCNC, Las Vegas, NV, USA, September 2006.

[113] D. N. Steven Aerts. *NxH1001 Coolflux Tranceiver Radio Specification.* 2007.

[114] *MSP430x1xx Family User's Guide.*

[115] *MSP430x161x datasheet.*

[116] *CC2538 SoC datasheet.* datasheet.

[117] *cc2538 SoC Family of Products.* datasheet.

[118] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis. *Weighted round-robin cell multiplexing in a general-purpose ATM switch chip.* IEEE Journal on Selected Areas in Communications, 9(8), October 1991.

[119] A. Demers, S. Keshav, and S. Shenker. *Analysis and Simulation of a Fair Queuing Algorithm.* In Journal of Internetworking: Research and Experience, 1:pp 3 – 26, September 1990.

[120] R. Aylward and J. A. Paradiso. *A Compact, High-Speed, Wearable Sensor Network for Biomotion Capture and Interactive Media.* In Proceedings of the 6th international conference on Information processing in sensor networks, 2007.

[121] R. Aylward and J. A. Paradiso. *Sensemble: A Wireless, Compact, Multi-User Sensor System for Interactive Dance.* In Proceedings of the conference on New interfaces for musical expression, 2006.

[122] O. Chipara, C. Lu, T. C. Bailey, and G.-C. Roman. *Reliable Clinical Monitoring using Wireless Sensor Networks: Experiences in a Step-down Hospital Unit.* In Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, SenSys, 2010.

[123] A. Rosi, N. Bicocchi, G. Castelli, M. Mamei, F. Zambonelli, M. Berti, and A. Corsini. *Landslide Monitoring with Sensor Networks: Experiences and Lessons Learnt from a Real-World Deployment.* International Journal of Sensor Networks, 10(3):pp 111 – 122, 2011.

[124] A. Wheeler. *Commercial Applications of Wireless Sensor Networks Using ZigBee.* IEEE Communications Magazine, 45(4):pp 70 – 77, 2007.

[125] S. J. Golestani. *A Stop-and-go Queueing Framework for Congestion Management.* ACM SIGCOMM Computer Communication Review, 20(4):pp 8 – 18, September 1990.

[126] L. Zhang. *Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks.* ACM SIGCOMM Computer Communication Review, 20(4):pp 19 – 29, September 1990.

[127] J. B. Nagle. *On Packet Switches with Infinite Storage.* IEEE Transactions on Communications, 35(4):pp 435 – 438, 1987.

[128] C. R. Kalmanek and H. Kanakia. *Rate Controlled Servers for Very High-Speed Networks.* In IEEE Global Telecommunications Conference, GLOBECOM '90, San Diego, CA, USA, December 1990.

[129] H. Zhang and S. Keshav. *Comparison of Rate-Based Service Disciplines.* ACM SIGCOMM Computer Communication Review, 21(4):pp 113 – 121, September 1991.

[130] A. Demers, S. Keshav, and S. Shenker. *Analysis and Simulation of a Fair Queuing Algorithm.* In SIGCOMM '89 Symposium proceedings on Communications architectures & protocols, USA, September 1989.

[131] A. Parekh and R. Gallager. *A generalized processor sharing approach to flow control in integrated services networks: the single node case.* In IEEE INFOCOM, 1992.

[132] A. Parekh and R. Gallager. *A generalized processor sharing approach to flow control in integrated services networks: the single node case.* IEEE Transactions on Networking, 1(3):pp 344 – 357, 1993.

[133] A. K. J. Parekh. *A Generalized Processor Sharing Approach to Flow Control In Integrated Services Networks.* Doctor of philosophy, Massachusetts Institute of Technology, 1992.

[134] J. S. Turner. *New Directions in Communications.* IEEE Communications Magazine, 24(10):pp 8 – 15, 1986.

[135] I. Stoica and H. A. Wahab. *Earliest Eligible Virtual Deadline First : A Flexible and Accurate Mechanism for Proportional Share Resource Allocation.* Technical Report TR95-22, Old Dominion University Norfolk, 1995.

[136] T. Ng, I. Stoica, and H. Zhang. *A hierarchical fair service curve algorithm for link sharing, Real-Time and Priority services.* In Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication, SIGCOMM, 1997.

[137] T. E. Ng, D. Stephens, I. Stoica, and H. Zhang. *Supporing Best-Effort Traffic with Fair Service Curve.* In Global Telecommunications Conference, GLOBECOM, Rio de Janeireo, December 1999.

[138] T. E. Ng, D. C. Stephens, I. Stoica, and H. Zhang. *Supporting Best-Effort Traffic with Fair Service Curve.* Technical Report CMU-CS-99-169, School of Computer Science, Carnegie Mellon University, Pittsburgh, 2000.

[139] K. Pyun. *A hierarchical Round-Robin Algorithm for Rate-Dependent Low Latency Bounds in Fixed-Sized Packet Networks.* Journal of Korean Institute of Information Scientists and Engineers, 32:pp 254 – 260, 2005.

[140] C. S. Chen and W. S. Wong. *Bandwidth allocation for wireless multimedia systems with most regular sequences.* IEEE Transactions on Wireless Communications, 4(2):pp 635 – 645, 2005.

[141] C. S. Chen and W. S. Wong. *Resource Sharing in the Most Regular Scheduling: Deterministic Performance and Guarantee.* In IEEE Military Communications Conference, MILCOM, 2006.

[142] B. Han, W. Jia, and L. Lin. *Performance evaluation of scheduling in IEEE 802.16 based wireless mesh networks.* Computer Communications, 30:pp 782 – 792, 2007.

[143] J. J. Babka. *System and method for fractional resource scheduling for video teleconferencing resources.* United States Patent, patent no: US 7,328,264 B2, Feb 2008.

[144] D. Ferrari, D. Verma, and H. Zhang. *Delay jitter control for real-time communication in a packet switching network.* In IEEE Conference on Communications Software, Communications for Distributed Applications and Systems, TRICOMM, 1991.

[145] D. Ferrari and D. C. Verma. *A Scheme for Real-Time Channel Establishment in Wide-Area Networks.* IEEE Journal on Selected Areas in Communications, 8:pp 368 – 379, 1990.

[146] M. Mock and E. Nett. *Real-Time Communication in Autonomous Robot Systems.* In The Fourth International Symposium on Autonomous Decentralized Systems, 1999. Integration of Heterogeneous Systems, 1999.

[147] M. Caccamo, L. Y. Zhang, L. Sha, and G. Buttazzo. *An implicit prioritized access protocol for wireless sensor networks.* In Real-Time Systems Symposium, RTSS, 2002.

[148] E. Egea-López, J. Vales-Alonso, A. S. Martínez-Sala, J. García-Haro, P. Pavón-Mariño, and M. V. Bueno-Delgado. *A Real-Time MAC Protocol for Wireless Sensor Networks: Virtual TDMA for Sensors (VTS).* Lecture Notes in Computer Science, Architecture of Computing Systems, 3894:pp 382 – 396, 2006.

[149] A. Rowe, R. Mangharam, and R. Rajkumar. *RT-Link: A Time-Synchronized Link Protocol for Energy-Constrained Multi-hopWireless Networks.* In IEEE Communications Society on Sensor and Ad Hoc Communications and Networks, SECON, 2006.

[150] *IEEE Standard for Local and metropolitan area networks- Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*, 2011. 802.15.4.

[151] J. Afonso, L. A. Rocha, H. R. Silva, and J. H. Correia. *MAC Protocol for Low-Power Real-Time Wireless Sensing and Actuation.* In IEEE International Conference on Electronics, Circuits and Systems, ICECS, 2006.

[152] Óscar Gama, P. Carvalho, J. A. Afonso, and P. M. Mendes. *An Improved MAC Protocol with a Reconfiguration Scheme for Wireless e-Health Systems Requiring Quality of Service.* In Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology, Wireless VITAE, 2009.

[153] J. A. Afonso, H. D. Silva, P. Macedo, and L. A. Rocha. *An Enhanced Reservation-Based MAC Protocol for IEEE 802.15.4 Networks.* Sensors, 11(4):pp 3852 – 3873, 2011.

[154] B. K. Singh and K. E. Tepe. *A Novel Real-time MAC Layer Protocol for Wireless Sensor Network Applications.* In IEEE International Conference on Electro/Information Technology, 2009.

[155] L. Choi, S. H. Lee, and J.-A. Jun. *SPEED-MAC: Speedy and Energy Efficient Data Delivery MAC Protocol for Real-Time Sensor Network Applications.* In IEEE International Conference on Communications (ICC), 2010.

[156] S. Ouni, S. Gherairi, and F. Kamoun. *Data aggregation and pipelining scheduling protocols for real-time wireless sensor networks.* International Journal of Ad Hoc and Ubiquitous Computing, 12(1):pp 56 – 64, 2013.

[157] V. Bharghavan, S. Lu, and R. Srikant. *Fluid flow fair scheduling emulation in wireless shared channel packet communication network*, April 2004.

[158] K. Gong. *Egyptian Fractions.* Technical report, UC Berkeley, 1992.

[159] D. Eppstein. *Algorithms for Egyptian Fractions.*

[160] *TmoteSky.* `http://www.snm.ethz.ch/Projects/TmoteSky`. Accessed March 3, 2010.

[161] *TinyOS Documentation Wiki.* `http://www.tinyos.net`. Accessed June 10, 2015.

[162] J. Snow, W. chi Feng, and W. chang Feng. *Implementing a Low Power TDMA Protocol Over 802.11.* In IEEE Wireless Communications and Networking Conference, WCNC, 2005.

[163] I. Hussain, N. Sarma, and D. K. Saikia. *TDMA MAC Protocols for WiFi-based Long Distance Networks: A Survey.* International Journal of Computer Applications, 94:pp 1 – 8, 2014. Published by Foundation of Computer Science, New York, USA.

[164] R. Moraes, F. Vasques, and P. Portugal. *A TDMA-based mechanism to enforce real-time behavior in WiFi networks.* In IEEE International Workshop on Factory Communication Systems, WFCS, pages pp 109 – 112, Dresden, Germany, May 2008.

[165] C. Ranveer, K. Sandeep, M. Thomas, N. Vishnu, P. Jitendra, R. Ramachandran, and R. Lenin. *DirCast: A Practical and Efficient Wi-Fi Multicast System.* In IEEE International Conference on Network Protocols,, Princeton, NJ, USA, October 2009.

[166] P. Bhagwat, B. Raman, and D. Sanghi. *Turning 802.11 inside-out.* ACM SIGCOMM Computer Communication Review, 34:pp 33 – 38, 2004.

[167] B. Raman and K. Chebrolu. *Design and evaluation of a new MAC protocol for long-distance 802.11 mesh networks.* In Annual International Conference on Mobile Computing and Networking, MobiCom, 2005.

[168] R. Patra, S. Nedevschi, S. Surana, A. Sheth, L. Subramanian, and E. Brewer. *WiLDNet: Design and Implementation of High Performance WiFi Based Long Distance Networks.* In USENIX Symposium on Networked Systems Design & Implementation, NSDI, Cambridge, MA, 2007.

[169] A. Dhekne, N. Uchat, and B. Raman. *Implementation and Evaluation of a TDMA MAC for WiFi-based Rural Mesh Networks.* In ACM Workshop on Networked Systems for Developing Regions, NSDR, 2009.

[170] Y. Ben-David, M. Vallentin, S. Fowler, and E. Brewer. *JaldiMAC: Taking the Distance Further.* In ACM Workshop on Networked Systems for Developing Regions, NSDR, USA, June 2010.

[171] Y. Ben-David, S. Fowler, and S. Hasan. *JaldiMAC – Beyond Simulation.* Technical Report CS262, Department of Electrical Engineering and Computer Science University of California, Berkeley, 2009.

[172] V. Sevani, B. Raman, and P. Joshi. *Implementation Based Evaluation of a Full-Fledged Multi-Hop TDMA-MAC for WiFi Mesh Networks.* IEEE Transactions on Mobile Computing, 13(2):pp 392 – 406, December 2012.

[173] P. Verkaik, Y. Agarwal, R. Gupta, and A. C. Snoeren. *Softspeak: Making VoIP Play Fair in Existing 802.11 Deployments.* In USENIX Symposium on Networked Systems Design and Implementation, NSDI, 2009.

[174] R. Costa, P. Portugal, F. Vasques, and R. Moraes. *A TDMA-based Mechanism for Real-Time Communication in IEEE 802.11e Networks.* In IEEE Conference on Emerging Technologies and Factory Automation, ETFA, Bilbao, Sept 2010.

[175] M. Neufeld, J. Fifield, C. Doerr, A. Sheth, and D. Grunwald. *SoftMAC - Flexible Wireless Research Platform.* In Hot Topics in Networks workshop, HotNets, College Park, MD, November 2005.

[176] A. Sharma, M. Tiwari, and H. Zheng. *MadMAC: Building a reconfigurable radio testbed using commodity 802.11 hardware.* In IEEE Workshop on Networking Technologies for Software Defined Radio Networks, WSDR, USA, September 2006.

[177] A. Sharma and E. M. Belding. *FreeMAC: framework for multi-channel mac development on 802.11 hardware.* In ACM workshop on Programmable routers for extensible services of tomorrow (ACM SIGCOMM PRESTO), August 2008.

[178] M.-H. Lu, P. Steenkiste, and T. Chen. *Using Commodity Hardware Platform to Develop and Evaluate CSMA Protocols.* In ACM international workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization, WiNTECH, 2008.

[179] P. Djukic and P. Mohaptra. *Soft-TDMAC: A Software TDMA-based MAC over Commodity 802.11 hardware.* In IEEE INFOCOM, Brazil, April 2009.

[180] P. Djukic and P. Mohaptra. *Soft-TDMAC: A Software-based 802.11 Overlay TDMA MAC with Microsecond Synchronization.* IEEE Transactions on Mobile Computing, 11:pp 478 – 491, 2012.

[181] V. Nikolyenko and L. Libman. *Coop80211: Implementation and Evaluation of a SoftMAC-based Linux Kernel Module for Cooperative Retransmission.* In IEEE Wireless Communications and Networking Conference, WCNC, 2011.

[182] P. Salvador, S. Paris, C. Pisa, P. Patras, Y. Grunenberger, X. Perez-Costa, and J. Gozdecki. *A modular, flexible and virtualizable framework for IEEE 802.11.* In Future Network & Mobile Summit, FutureNetw, 2012.

[183] M. E. Mosko. *TDMA communication using a CSMA chipset*, June 2013.

[184] Y.-H. Wei, Q. Lengy, S. Hany, A. K. Moky, W. Zhangz, and M. Tomizuka. *RT-WiFi: Real-Time High-Speed Communication Protocol for Wireless Cyber-Physical Control Applications.* In IEEE Real-Time Systems Symposium, RTSS, Canada, December 2013.

[185] G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, F. Gringoli, and I. Tinnirello. *MAClets: Active MAC Protocols over Hard-Coded Devices.* In International Conference on emerging Networking EXperiments and Technologies, CoNEXT, France, December 2012.

[186] T. Gleixner and D. Niehaus. *Hrtimers and beyond: transforming the Linux time subsystems.* In Linux Symposium, volume 1, Canada, July 2006.

[187] K.-D. Kwon, M. Sugaya, and T. Nakajima. *KTAS: Analysis of Timer Latency for Embedded Linux Kernel.* International Journal of Advanced Science and Technology, 19:pp 59 – 70, 2010.

[188] S. A. Rodriguez and P. M. S. Burt. *A Latency Model of Linux 2.6 for Digital Signal Processing in Real Time.* In Twelfth Real-Time Linux Workshop, Nairobi, Kenya, 2010.

[189] D. P. Bovet and M. Cesati. *Understanding the Linux Kernel, 3rd edition.* O'Reilly Media, Sebastopol, CA 95472, 2005. ISBN 978-0-596-00565-8.

[190] R. Love. *Linux Kernel Development, 2nd Edition.* Novell Press, United States of America, 2005. ISBN 978-0-672-32720-9.

[191] R. Love. *Linux Kernel Development, 3rd Edition.* Addison-Wesley Professional, 2010. ISBN 978-0-672-32946-3.

[192] *The Linux Kernel source.* `www.kernel.org`.

[193] M. Jones. *Inside the Linux 2.6 Completely Fair Scheduler.* Technical report, IBM Corporation, 2009.

[194] J. Corbet, G. Kroah-Hartman, and A. Rubini. *Linux Device Drivers, 3rd edition.* O'Reilly Media, Sebastopol, CA 95472, 2005. ISBN 978-0-596-00590-0.

[195] *The Impact Of A Tickless Kernel.* `http://www.phoronix.com/scan.php?page=article&item=651&num=1`. Accessed April 8, 2014.

[196] *High resolution timers and dynamic ticks design notes.* https://www.kernel.org/doc/Documentation/timers/highres.txt. Accessed April 8, 2014.

[197] *IBM Timers in the 2.6 kernel.* http://www.ibm.com/developerworks/library/l-timers-list/. Accessed September 3, 2013.

[198] I. Molnar. *Ingo Molnar on timer.c design.* http://lwn.net/Articles/156329/. Accessed Dec 9, 2013.

[199] *High Resolution Timers.* https://rt.wiki.kernel.org/index.php/High_resolution_timers. Accessed December 19, 2014.

[200] *Real-Time Linux Wiki.* https://rt.wiki.kernel.org/index.php/Main_Page. Accessed September 4, 2013.

[201] S. Venkateswaran. *Essential Linux Device Drivers.* Prentice Hall Open Source Software Development Series. Prentice Hall, Boston, MA, USA, 2008. ISBN 978-0-132-39655-4.

[202] C. Benvenuti. *Understanding Linux Network Internals.* O'Reilly Media, Sebastopol, CA 95472, 2005. ISBN 978-0-596-10367-5.

[203] R. Rosen. *Linux Kernel Networking: Implementation and Theory.* Apress, 2013. ISBN 978-1-4302-6196-4.

[204] *Linux Wireless wiki.* https://wireless.wiki.kernel.org/. Accessed January 30, 2015.

[205] B. S. I. G. (SIG). *Bluetooth Network Encapsulation Protocol (BNEP) Specification*, February 2003. Specification of the Bluetooth System, Version 1.0.

[206] *netlink(7) - Linux man page.* http://linux.die.net/man/7/netlink. Accessed January 9, 2014.

[207] G. C. Buttazzo. *Hard Real-Time computing systems: predictable scheduling algorithms and applications.* Kluwer Academic Publishers Group, 3300 AH Dordrecht, The Netherlands, 1960.

[208] A. Bastoni. *Towards the Integration of Theory and Practice in Multiprocessor Real-Time Scheduling.* PhD thesis, Universita Degli Studi Di Roma "Tor Vergata", 2011.

[209] M. T. Jones. *Anatomy of real-time Linux architectures: From soft to hard real-time.* Technical report, Emulex Corp., 2008.

[210] K. Yaghmour, J. Masters, G. Ben-Yossef, and P. Gerum. *Building Embedded Linux Systems.* O'Reilly Media, Sebastopol, CA 95472, 2008. ISBN 978-0-596-52968-0.

[211] *Xenomai.* http://www.xenomai.org/. Accessed December 8, 2013.

[212] P. Gerum. *LWN.net I-pipe: Introduction.* `http://lwn.net/Articles/140374/`. Accessed Feb 17, 2015.

[213] K. Yaghmour. *Adaptive Domain Environment for Operating Systems.* whitepaper, 2001.

[214] Free Electrons, Avignon. *Embedded Linux system development training course*, November 21-25 2011.

[215] *eCos.* `http://ecos.sourceware.org/`. Accessed September, 2013.

[216] A. J. Massa. *Embedded software development with eCos.* Pearson Education, Inc., Upper Saddle River, New Jersey 07458, 2003. ISBN 0-13-035473-2.

[217] E. Rudolph, Y. Rui, H. S. Malvar, L. wei He, M. F. Cohen, and I. Tashev. *Systems and methods for real-time audio-visual communication and data collaboration in a network conference environment*, December 2009.

[218] T. Bird, W. Newton, and J. lagnel. *Kernel Timer Systems*, 2017.

[219] *cyclictest RT wiki.* `https://rt.wiki.kernel.org/index.php/Cyclictest`. Accessed September 2, 2013.

[220] *AR9280 Product Brief.* `http://www.wikidevi.com/files/Atheros/specsheets/AR9280_1.pdf`.

[221] *AR9002U Product Brief.* `https://wikidevi.com/files/Atheros/specsheets/AR7010+AR9280.pdf`.

[222] Atheros. *AR9280 Single-Chip 2x2 MIMO MAC/BB/Radio datasheet.* 2009.

[223] S. Leffler. *TDMA for Long Distance Wireless Networks.* In Computer Laboratory Seminar, University of Cambridge, Cambridge, September 2009.

[224] A. M. Wyglinski, M. Nekovee, and Y. T. Hou. *Cognitive Radio Communications and Networks: Principles and Practice.* Academic Press, 2010. ISBN 978-0-12-374715-0.

[225] J. Mitola. *Cognitive Radio, An Integrated Agent Architecture for Software Defined Radio.* PhD thesis, Royal Institute of Technology (KTH), Kista, Sweden, 2000.

[226] I. F. Akyildiz, W.-Y. Lee, and K. R. Chowdhury. *CRAHNs: Cognitive radio ad hoc networks.* Ad Hoc Networks, 7(5):pp 810 – 836, 2009.

[227] Y. Xiao and F. Hu. *Cognitive Radio Networks.* Auerbach Publications, 2009. ISBN 978-1-4200-6420-9.

[228] J. Mo, H.-S. So, and J. C. Walrand. *Comparison of multichannel MAC protocols.* IEEE Transactions on Mobile Computing, 7(1):pp 50 – 65, January 2008.

[229] A. D. Domenico, E. C. Strinati, and M.-G. D. Benedetto. *A Survey on MAC Strategies for Cognitive Radio Networks.* IEEE Communications Surveys & Tutorials, 14(1):pp 21 – 44, 2012.

[230] N. Nguyen-Thanh, A. T. Pham, and V.-T. Nguyen. *Medium Access Control Design for Cognitive Radio Networks: A Survey.* IEICE Trans. Communications, Special Section on Technologies for Effective Utilization of Spectrum White Space, E97-B(2):pp 1 – 18, 2014.

[231] J. D. Dunagan, P. Bahl, and R. Chandra. *Slotted seeded channel hopping for capacity improvement in wireless networks*, May 2008.

[232] K. Bian, J.-M. Park, and R. Chen. *A quorum-based framework for establishing control channels in dynamic spectrum access networks.* In 15th annual international conference on Mobile computing and networking (MobiCom), Beijing, China, 20-25 September 2009.

[233] E. Lee, S. Oh, and M. Gerla. *Randomized channel hopping scheme for anti-jamming communication.* In IFIP Wireless Days (WD), Venice, Italy, 20-22 October 2010.

[234] M. Altamini, K. Naik, and X. Shen. *Parallel link rendezvous in ad hoc cognitive radio networks.* In IEEE Global Telecommunications Conference (GLOBECOM), Miami, FL, 6-10 December 2010.

[235] S. Romaszko. *Making a Blind Date the Guaranteed Rendezvous in Cognitive Radio Ad Hoc Networks.* In European Wireless Conference (EW), number ISBN 978-3-8008-3426-9, Poznan, Poland, 18-20 April 2012. l' VDE VERLAG GMBH.

[236] S. Romaszko and P. Mähönen. *Quorum systems towards an asynchronous communication in Cognitive Radio Networks.* Journal of Electrical and Computer Engineering (JECE), 2012:pp 1 – 22, 2012.

[237] M. Vukolic. *The Origin of Quorum Systems.* Bulletin of the EATCS 101: 125-147, 2010.

[238] S. Romaszko and P. MÍahÍonen. *Asynchronous channel allocation in opportunistic cognitive radio networks.* IGI global, 2014.

[239] M. Maekawa. *A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems.* ACM Transactions on Computer Systems (TOCS), 3(2):pp 145 – 159, 1985.

[240] C.-M. Chao, J.-P. Sheu, and I.-C. Chou. *An Adaptive Quorum-Based Energy Conserving Protocol for IEEE 802.11 Ad Hoc Networks.* IEEE Transaction on Mobile Computing, 5:pp 560 – 570, 2006.

[241] S. Lang and L. Mao. *A torus quorum protocol for distributed mutual exclusion.* In International Conference on Parallel and distributed computing and systems (PDCS), pages pp 635 – 638, Las Vegas, USA, 1-4 October 1998.

[242] J. M. Hall. *Combinatorial Theory.* John Wiley and Sons, New York, 1986.

[243] W. S. Luk and T. T. Wong. *Two new quorum based algorithms for distributed mutual exclusion.* In 17th International Conference on Distributed Computing Systems (ICDCS), May 1997.

[244] A. Bogomolny. *Interactive Mathematics Miscellany and Puzzles from Interactive Mathematics Miscellany and Puzzles.* `http://www.cut-the-knot.org/arithmetic/latin_intro.shtml`. Accessed 19 September 2014.

[245] K. Bian, J.-M. Park, and R. Chen. *Control Channel Establishment in Cognitive Radio Networks using Channel Hopping.* IEEE Journal on Selected Areas in Communications, 29(4):pp 689 – 703, 2011.

[246] Y. Zhang, G. Yu, Q. Li, H. Wang, X. Zhu, , and B. Wang. *Channel Hopping Based Communication Rendezvous in Cognitive Radio Networks.* IEEE/ACM Transactions on Networking, 22(3):pp 889 – 902, 2014.

[247] S. Romaszko, W. Torfs, P. Mähönen, and C. Blondia. *How to be an efficient asynchronous neighbourhood discovery protocol in opportunistic cognitive wireless networks.* International Journal of Ad Hoc and Ubiquitous Computing, 20(3), 2015.

[248] S. Romaszko, D. Denkovski, V. Pavlovska, and L. Gavrilovska. *Quorum system and random based asynchronous rendezvous protocol for cognitive radio ad hoc networks.* EAI Endorsed Transactions on Mobile Communications and Applications, 13(3):pp 1 – 14, 2013.

[249] Z. Lin, H. Liu, X. Chu, , and Y.-W. Leung. *Enhanced Jump-Stay Rendezvous Algorithm for Cognitive Radio Networks.* IEEE Communication Letters, 17(9):pp 1742 – 1745, 2013.

[250] S. Romaszko, W. Torfs, P. Mähönen, and C. Blondia. *Benefiting from an Induced Asynchronism in Neighborhood Discovery in opportunistic Cognitive Wireless Networks.* In ACM International Symposium on Mobility Management and Wireless Access (MobiWac), Barcelona, Spain, 3-4 November 2013.

[251] S. Romaszko, W. Torfs, P. Mähönen, and C. Blondia. *AND: Asynchronous Neighborhood Discovery protocols for opportunistic Cognitive Wireless Networks.* In IEEE Wireless Communications and Networking Conference (WCNC), Istanbul, Turkey, 6-9 April 2014.

[252] P. Bahl, R. Chandra, and J. Dunag. *SSCH: Slotted Seeded Channel Hopping for Capacity Improvement in IEEE 802.11 Ad Hoc Wireless Networks.* In Annual International Conference on Mobile Computing and Networking (MOBICOM), pages pp 216 – 230, 2004.

[253] K. Balachandran and J. Kang. *Neighbor discovery with dynamic spectrum access in adhoc networks.* In IEEE 63rd Vehicular Technology Conference (VTC-Spring), volume 2, pages pp 512 – 517, May 2006.

[254] W. Hung, D. Willkomm, M. Abusubaih, J. Gross, G. Vlantis, M. Gerla, and A. Wolisz. *Dynamic Frequency Hopping Communities for efficient IEEE 802.22 operation.* IEEE Communication Magazine, 45(5):pp 80 – 87, 2007.

[255] C. Cordeiro and K. S. Challapali. *C-MAC: a cognitive MAC protocol for multi-channel wireless networks.* In in Proceedings of the 2nd International Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN), Dublin, Ireland, 2007.

[256] H. Su and X. Zhang. *Channel-hopping based single transceiver MAC for cognitive radio networks.* In Conference on Information Sciences and Systems (CISS), pages pp 197 – 202, Princeton, New Jersey, USA, 19-21 March 2008.

[257] L. Jiao and F. Li. *A Single radio based channel datarate-aware parallel rendezvous MAC protocol for cognitive radio networks.* In IEEE Conference on Local Computer Networks (LCN), pages pp 392 – 399, Zurich, Switzerland, 20-23 October 2009.

[258] C.-M. Chao and H.-C. Tsai. *A Channel Hopping Multi-Channel MAC Protocol for Mobile Ad Hoc Networks.* IEEE Transactions on Vehicular Technology, November 2009.

[259] C.-M. Chao, H.-C. Tsai, and K.-J. Huang;. *A new channel hopping MAC protocol for mobile ad hoc networks.* In Wireless Communications & Signal Processing (WCSP), Nanjing, China, 13-15 November 2009.

[260] C. Cormio and K. Chowdhury. *An adaptive multiple rendezvous control channel for Cognitive Radio wireless ad hoc networks.* In IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), pages pp 346 – 351, Mannheim, Germany, 29 March- 2 April 2010.

[261] C. Cormio and K. Chowdhury. *Common control channel design for cognitive radio wireless ad hoc networks using adaptive frequency hopping.* Elsevier Ad Hoc Networks, 8:pp 430 – 438, 2010.

[262] L. DaSilva. and I. Guerreiro. *Sequence-Based Rendezvous for Dynamic Spectrum Access.* In IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN), Chicago, Illinois, USA, 14-17 October 2008.

[263] N. C. Theis, R. W. Thomas, , and L. A. DaSilva. *Rendezvous for Cognitive Radios.* IEEE Transactions on Mobile Computing, 10:pp 216 – 227, 2010.

[264] Y.-S. Hsieh, C.-W. Lien, and C.-T. Chou. *A multi-channel testbed for dynamic spectrum access (DSA) networks.* In ACM Workshop on Wireless Multimedia Networking and Computing (WMuNeP), pages pp 63 – 66, Miami, Florida, USA, 31 October - 04 November 2011.

[265] Y. Zhang, Q.Li, G.Yu, and B. Wang. *ETCH: Efficient Channel Hopping for Communication Rendezvous in Dynamic Spectrum Access Networks.* In IEEE International Conference on Computer Communications (INFOCOM), pages pp 2471 – 2479, Shanghai, China, 10-15 April 2011.

[266] Z. Gu, Q.-S. Hua, Y. Wang, and F. C. M. Lau. *Nearly Optimal Asynchronous Blind Rendezvous Algorithm for Cognitive Radio Networks.* In IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON), pages pp 371 – 379, New Orleans, USA, 24-27 June 2013.

[267] Z. Lin, H. Liu, X. Chu, and Y. W. Leung. *Jump-Stay based channel-hopping algorithm with guaranteed rendezvous for cognitive radio networks.* In IEEE International Conference on Computer Communications (INFOCOM), pages pp 2444 – 2452, Shanghai, China, 10-15 April 2011.

[268] H. Liu, Z. Lin, X. Chu, and Y.-W. Leung. *Jump-Stay Rendezvous Algorithm for Cognitive Radio Networks.* IEEE Transactions on Parallel and Distributed Systems, 23(10):pp 1867 – 1881, 2012.

[269] R. Gandhi, C.-C. Wang, and Y. C. Hu. *Fast rendezvous for multiple clients for cognitive radios using coordinated channel hopping.* In IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON), pages pp 434 – 442, Seul, Korea, June 2012.

[270] H. Liu, Z. Lin, X. Chu, and Y.-W. Leung. *Ring-Walk Based Channel-Hopping Algorithms with Guaranteed Rendezvous for Cognitive Radio Networks.* In IEEE/ACM International Conference on Cyber, Physical and Social Computing (CPSCom), Green Computing and Communications (GreenCom), pages pp 755 – 760, Hangzhou, China, 18-20 December 2010.

[271] G.-Y. Chang, W.-H. Teng, H.-Y. Chen, and J.-P. Sheu. *Novel Channel-Hopping Schemes for Cognitive Radio Networks.* IEEE Transactions on Mobile Computing, 13:pp 407 – 421, 2014.

[272] K. Bian and J.-M. Park. *Asynchronous Channel Hopping for Establishing Rendezvous in Cognitive Radio Networks.* In IEEE International Conference on Computer Communications (INFOCOM), Mini-Conference, pages pp 236 – 240, Shanghai, China, 10-15 April 2011.

[273] K. Bian and J.-M. Park. *Maximizing Rendezvous Diversity in Rendezvous Protocols for Decentralized Cognitive Radio Networks.* IEEE Transactions on Mobile Computing, 12(7):pp 1294 – 1307, July 2012.

[274] F. Hou, L. Cai, X. Shen, , and J. Huang. *Asynchronous Multichannel MAC Design With Difference-Set-Based Hopping Sequences.* IEEE Transactions on Vehicular Technology, 60(4):pp 1728 – 1739, 2011.

[275] S. Romaszko and P. Mähönen. *A Rendezvous protocol with the heterogeneous spectrum availability analysis for Cognitive Radio Ad Hoc Networks.* Hindawi Journal of Electrical and Computer Engineering (JECE), Special Issue on Resource Allocation, 2013:pp 1 – 18, 2012.

[276] S. Romaszko, W. Torfs, P. Mähönen, and C. Blondia. *An analysis of asynchronism of a neighborhood discovery protocol for cognitive radio networks.* In IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), pages pp 3155 – 3160, London, UK, 8-11 September 2013.

[277] A. Robertson, L. Tran, J. Molnar, and E.-H. F. Fu. *Experimental comparison of blind rendezvous algorithms for tactical networks.* In IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), pages pp 1 – 6, San Francisco, USA, 25-28 June 2012.

[278] S. Romaszko and P. Mähönen. *Torus Quorum System and Difference Set-based Rendezvous in Cognitive Radio Ad Hoc Networks.* In International Conference on Cognitive Radio Oriented Wireless Networks and Communications (CrownCom), Stockholm, Sweden, June 2012.