University of Antwerp
Faculty of Sciences
Department of Mathematics &
Computer Science

Network-aware resource allocation algorithms for service orchestration in heterogeneous cloud environments

Bart Spinnewyn

Thesis submitted for the degree of doctor in Computer Science at the University of Antwerp, to be defended by Bart Spinnewyn, Academic year 2018-2019.

University of Antwerp
Faculty of Sciences
Department of Mathematics &
Computer Science

Promotors: Prof. Dr. Steven Latré
Prof. Dr. Juan Felipe Botero Vega

University of Antwerp
Faculty of Sciences

Department of Mathematics & Computer Science
Sint-Pietersvliet 7, B-2000 Antwerp, Belgium

Thesis submitted for the degree of doctor in
Computer Science at the University of
Antwerp, to be defended by Bart Spinnewyn,
Academic year 2018-2019.

Nederlandse samenvatting – Summary in Dutch

De volgende generatie van Internetdiensten, zoals, zelfrijdende auto's, geaugmenteerde werkelijkheid en slimme robots waarvoor de intelligentie in de cloud zit, vereisen draadloze communicatiemiddelen die zeer weinig vertraging introduceren; produceren gigantische hoeveelheden data en vereisen dat het opzetten van digitale samenwerkingen tussen uiteenlopende apparaten gezwind gebeurt. Terwijl een traditionele clouddienst typisch wordt opgezet binnen één enkele server cluster, hebben deze nieuwe diensten bandbreedte en reactietijd vereisten die opleggen dat minstens enkele computationele taken uitgevoerd worden nabij de eindgebruiker. In een eerste fase van ruimtelijke distributie van de cloud-infrastructuur verbinden Internetdiensten cloud-elementen tussen meerdere clusters van servers. In een tweede fase worden cloud-elementen nog dichter geplaatst bij de eindgebruiker. Tegenwoordig, naar het paradigma van edge-computing, worden rekencapaciteiten geplaatst aan de rand van het netwerk. Binnen het paradigma van mist- en fog-computing worden deze elementen zelfs binnen het (draadloze) toegangsnetwerk geplaatst, om de afstand tot de eindgebruiker verder te minimaliseren. Om de technische en economische haalbaarheid van deze nieuwe internettoepassingen te verzekeren, is het vermogen om op een effectieve en vraag-gedreven wijze, de reken-, netwerk-, en opslagcapaciteiten aan de rand van het netwerk te orkestreren, onontbeerlijk.

De inlijving van perifere apparatuur in de cloud-omgeving, dewelke beperkte en ongelijkmatig verdeelde capaciteiten kent en minder betrouwbaar is, brengt serieuze uitdagingen met zich mee i.v.m. het beheer. Huidige algoritmes voor de middelentoewijzing in cloud-systemen kunnen niet overweg met de verhoogde schaarste en onbetrouwbaarheid van middelen aan de rand van het netwerk. In dit proefschrift identificeren we de uitdagingen waarmee ruimtelijk gedistribueerde cloud-omgevingen te kampen hebben en reiken we technieken en algoritmes aan om deze uitdagingen weg te werken. Ten eerste zijn er uitdagingen betreffende de vermenigvuldiging van gegevens over meerdere opslagvoorzieningen aan de rand van het netwerk. Replicatie van gegevens is een techniek die traditioneel aangewend wordt om zeer betrouwbare gegevensopslag te realiseren d.m.v. standaard, commerciële computerapparatuur. In gecentraliseerde server clusters kan foutloze gegevensopslag op deze wijze verzekerd worden gedurende duizenden jaren. Deze lange beschikbaarheid is mogelijk omdat de nagenoeg onbegrensde opslagcapaciteiten binnen een server cluster toestaan dat de gegevens uitvoerig gerepliceerd worden. Aan de rand van het netwerk is de kans op faling echter meer uitgesproken en zijn opslagcapaciteiten veel schaarser. Daarom is opslag aan de rand van het

netwerk hoofdzakelijk bestemd voor tijdelijke bewaring van gegevens. Als een gevolg daarvan zijn slimme algoritmes nodig om de replicatie van gegevens te sturen, die aangepast aan de vereiste kwaliteit van dienstverlening (SLA) die met de gegevens gepaard gaat en aan de infrastructuur waarop de gegevens bewaard worden. Daarom ontwikkelen we een SLA-bewust replicatie algoritme dat dynamisch het verwachte gegevensverlies en de opslagkosten kan balanceren over de levensduur van een dienst. D.m.v. uitgebreide simulaties tonen we aan dat onze aanpak de winstgevendheid van gegevensopslag kan verbeteren over een breed gamma aan SLA- en cloud-omstandigheden en de replicatie-strategie kan aanpassen aan veranderende operationele omstandigheden.

Ten tweede zijn er uitdagingen om betrouwbare netwerkdiensten op te zetten die een combinatie van node- en link-falen kunnen weerstaan. De schaarste aan middelen en het verhoogde faalgedrag compliceren de middelentoewijzing aan de rand van het netwerk voor virtuele netwerk allocaties die falingen in de onderliggende infrastructuur dienen te weerstaan (SVNEs). Huidige algoritmes kunnen niet overweg met de verminderde uniformiteit, gezien deze een vast niveau van bescherming introduceren. Daarom stellen we een methode voor om SVNEs te alloceren die zowel de beoogde beschikbaarheid van de dienst in overweging neemt als beschermingen enkel introduceert daar waar ze nodig zijn om een minimale beschikbaarheid voor de gehele dienst te realiseren, en dit met minimale kost. Een uitgebreide evaluatie toont aan dat vergeleken met algoritmes die een vast beschermingsniveau voorzien, onze voorgestelde heuristiek het aantal applicaties waarvan de beschikbaarheidsvereisten voldaan zijn, kan verdubbelen in cloud-omgevingen met meer dan 100 nodes, terwijl de vereiste rekentijd om een oplossing te vinden beperkt blijft tot 20 seconden.

Ten derde zijn er uitdagingen gerelateerd aan de orkestratie van netwerkdiensten op basis van een combinatie van functionele en niet-functionele vereisten, wanneer de infrastructuur sterk niet-uniform is. Er is nood aan orkestratie algoritmes die een dienst kunnen samenstellen uit virtuele netwerkfuncties (VNFs), op basis van zijn vereisten m.b.t. de geleverde dienst en met oog voor de beperkingen van de gedeelde middelen aan de rand van het netwerk (MEC). Huidige modellen voor de compositie van dergelijke diensten tot een VNF-FG, beperken gewoonlijk de samenstelling van deze logische structuur van de dienst tot VNFs die verbonden zijn in een pad- of boomstructuur. Gezien de structuur van diensten in realiteit vele malen complexer is, poneren we een nieuw model met bredere toepasbaarheid dat de VNFs kan verbinden met de topologie van een gerichte acyclische graaf. Verder, terwijl huidige orkestratie aanpakken gewoonlijk de compositie van de VNF-FG en de toewijzing van deze gevirtualiseerde middelen aan fysieke bronnen in het netwerk typisch uitvoeren met weinig tot geen coördinatie tussen beide taken, stellen wij algoritmes voor die compositie en toewijzing uitvoeren op een gecoördineerde manier. Experimenten tonen aan dat coördinatie van beide taken het aantal diensten die gelijktijdig gebruik kunnen maken van de infrastructuur gevoelig kan verhogen.

Samengevat, gecentraliseerde en ruimtelijk gedistribueerde cloud-omgevingen verschillen sterk. In vergelijking tot gecentraliseerde clouds zijn gedistribueerde omgevingen veel minder uniform. Gedistribueerde omgevingen incorporeren namelijk zowel de infrastructuur van gecentraliseerde omgevingen, als de infrastructuur aan de rand van het netwerk, welke veel minder betrouwbaar is en vaak beperkte capaciteiten heeft. Deze kloof in betrouwbaarheid, connectiviteit, en afstand tot de eindgebruiker, bemoeilijkt het beheer van dergelijke omgevingen aanzienlijk. Dit proefschrift onderzoekt de uitdagingen die gepaard gaan met de orkestratie van Internetdiensten in deze heterogene cloud-omgevingen en reikt technieken aan om deze uitdagingen aan te gaan. Ten eerste, onderzoeken we hoe gegevens op een effectieve wijze gerepliceerd kunnen worden over opslagvoorzieningen aan de rand van het netwerk. We maximaliseren de opbrengsten van de netwerkbeheerder in situ over de levensduur van

een dienst. Deze optimalisatie beschouwt zowel SLAs betreffende beschikbaarheid en de cloud-karacteristieken. Deze aanpak is gestoeld op een dynamisch falingsmodel dat zowel de impact van faalgedrag en de benodigde tijd om data te repliceren beschouwt. Ten tweede, onderzoeken we hoe betrouwbare geheugenloze netwerkdiensten kunnen opgezet worden in een omgeving waar zowel apparaten als hun verbindingen falen, waarbij een minimale beschikbaarheid gegarandeerd wordt voor elke dienst en de plaatsingskost geminimaliseerd wordt. Om de beperkingen van de infrastructuur aan de rand van het netwerk te omzeilen introduceert onze SVNE-techniek, die zich bewust is van de beschikbaarheidsvereisten, enkel beschermingen waar deze nodig zijn. Ten derde, onderzoeken we hoe een netwerkdienst te orkestreren in een NFV-omgeving. We stellen algoritmes voor die het aantal diensten die tegelijkertijd eenzelfde infrastructuur kunnen delen; en de kwaliteit van de configuraties, kunnen verbeteren door coördinatie van de compositie van middelen, enerzijds; en de toewijzing van deze middelen aan bronnen in de fysieke infrastructuur, anderzijds. Naast de ontwikkeling van de vereiste middelentoewijzingsalgoritmes voor een bestaand model van netwerkdiensten met een boomstructuur, ontwikkelen we ook een nieuw model met bredere toepasbaarheid en ook de vereiste algoritmes.

English summary

The next generation of Internet services, e.g., self-driving cars, Cloud-Radio Access Network (C-RAN), ubiquitous and remote sensing, Augmented Reality (AR) and cloud robotics, requires ultra-low latency wireless communications, produces vast quantities of data and requires the speedy deployment of real-time collaborations between a wide variety of devices. While traditionally cloud services are hosted on infrastructure that is located within a single datacenter, these novel services have throughput and response time requirements that dictate that at least some computational tasks are executed near the location of the end user. In a first phase of geo-distribution of the cloud infrastructure, services interconnect cloud elements across multiple datacenters. In a second phase of geo-distribution, cloud elements are moved even closer to the end-user. Now, computational elements are being placed at the edge of the network, in what is referred to as edge computing, or even within the (wireless) Local Area Network (LAN), in mist and fog computing. For these applications to be at the same time technically feasible and economically viable, they will depend on the ability to effectively orchestrate computing, networking and storage resources at the network edge, on-demand.

Incorporating devices at the network edge that have limited and diverse capabilities and an increased failure probability, poses severe challenges with regard to the manageability of the infrastructure. Current cloud resource allocation algorithms cannot deal in a meaningful way with the resource scarcity and best-effort behavior at the edge. In this monograph, we identify the orchestration challenges related to these geo-distributed deployments and propose mechanisms and algorithms to mitigate those challenges.

First, there are challenges related to replication of application data across multiple edge nodes. Data replication traditionally enables highly available cloud storage on top of commodity hardware. In datacenters, the Mean Time to Dataloss (MTDL) is in the order of thousands of years and the storage capabilities are virtually infinite. At the network edge however, failure is much more present and storage capabilities are severely reduced. Hence, storage at the network edge should be temporary. Consequently, intelligent replication algorithms are needed that are optimized for the required Quality of Service (QoS)-level and that consider the infrastructure on which the services are deployed. Hence, we propose a Service Level Agreement (SLA)-aware replication algorithm that dynamically balance data loss and storage cost over the service lifetime. Through extensive simulations, we show that our approach significantly improves the provider revenue over a wide range of cloud- and SLA-conditions and adapts its strategy to evolving operating conditions.

Second, there are challenges related to deploying reliable Network Services (NSs) that can survive a combination of node and link failures. Given the scarcity of resources and the

increased failure probability, provisioning Survivable Virtual Network Embeddings (SVNEs) at the edge is very challenging. Current SVNE algorithms cannot deal with the increased heterogeneity as they introduce a fixed level of redundancy to protect the NS. Therefore, we propose availability-aware SVNE mechanisms and resource allocation algorithms that introduce protections only where they are needed in order to realize a minimum degree of availability for the entire service at minimal cost. A detailed performance evaluation shows that compared to algorithms that produce SVNEs with a fixed protection level, our proposed heuristic can double the number of applications satisfying availability requirements, in cloud environments comprising over 100 nodes, while keeping the time required to calculate the solution under 20 seconds.

Third, there are challenges related to the orchestration of NSs based on their service requirements when the infrastructure is highly heterogeneous. Service orchestration algorithms are needed that can compose the service based on the service requirements while considering the Multi-access Edge Computing (MEC) resources in the Network Functions Virtualization (NFV) environment. Current service models typically considered that the logical structure of the service is either an Service Function Chain (SFC) or a tree. Hence, we propose a novel service model with wider applicability that can be used to compose Virtual Network Functions (VNFs) into a Directed Acyclic Graph (DAG) VNF-Forwarding Graph (VNF-FG). Further, while current orchestration approaches typically performed composition and embedding with poor coordination between these two tasks, we propose algorithms that can compose and embed such services in a coordinated way. Numerical experiments show that, through coordination of both tasks, our proposed algorithms can significantly improve the acceptance ratio and reduce the embedding cost, compared to algorithms that perform these tasks in two uncoordinated stages.

Summarized, centralized and geo-distributed cloud environments are worlds apart. Compared to centralized clouds, geo-distributed cloud environments are much more heterogeneous. These environments incorporate both infrastructure in datacenters and infrastructure at the network edge with very limited capabilities and that is much more failure-prone. This spread on capability; reliability; connectivity; and proximity to the end-user, severely complicates the management. This thesis investigates the challenges related to the orchestration of NSs across heterogeneous cloud environments and proposes novel management approaches that address these challenges. First, we investigate how to effectively replicate data across storage nodes in these environments. We approach this problem as a runtime revenue problem, that considers both SLAs regarding durability and the cloud characteristics. This approach builds on a dynamic availability model that considers both the impact of failure distribution and recovery times on data loss. Second, we investigate how to protect stateless NSs against a combination of node and link failure in these environments. We approach the problem of placing applications, while guaranteeing a minimum availability for each application and minimizing the placement cost as a resource allocation problem. To deal with the scarcity of resources at the edge and the reliability spread, our availability-aware approach introduces protections only where they are needed. Third, we investigate how to orchestrate NSs in an NFV environment. We propose orchestration algorithms that can improve the acceptance ratio and placement quality through coordination of the service composition and embedding. Not only do we develop the required orchestration algorithms for an existing service model that can generate VNF-FGs with a tree topology, we also develop a novel service model with improved applicability and develop the required orchestration algorithms.

Acknowledgements

Firstly, I would like to thank my advisor Prof. Steven Latré for the support of my PhD study and related research. He introduced me to the wonderful world of cloud computing and virtualization, which is simple and pure. The past four years have been a great journey, where I had the privilege to research the topics that interested me most.

Besides my advisor, I would like to express my sincere gratitude to Prof. Juan Felipe Botero Vega of the University of Antioquia for his guidance and constructive feedback to my work. Further, I thank Carlos Donato for his valuable contributions to the 5G and Twitch media-streaming use-cases.

I thank my fellow PhD students for the stimulating discussions, for all the fun we have had in the last four years, and lending a hand in times of need.

Also, I thank my friends at Applied Telecommunication Research Group (GITA) of the University of Antioquia, for welcoming me in your beautiful country and supporting me throughout my 6 month stay. Further, I thank Joost Bosman, Prof. Hans van den Berg and Prof. Rob van der Mei for the stimulating discussions we had during my research stay at the Stochastics group at Center for Mathematics and Computer Science (CWI) in Amsterdam, Netherlands.

Finally, I would like to thank my friends and family for their continued support throughout the entirety of my studies. I am forever indebted to my parents and brothers, for being great role models and providing me with a loving home-environment. Special thanks go out to Gert-Jan Stockman for his friendship, practical advice and proofreading of my manuscripts. I am indebted to my friend Stijn Vanhamel, for being there whenever I need him and helping me distract from my research at times. Last but not least, I thank my girlfriend Jeanny Bosack for her love and support throughout my life and teaching me the things that one cannot learn from books.

Antwerp, March 2019
Bart Spinnewyn

Table of Contents

List of Tables

List of Figures

List of Acronyms

5GUARDS	5G qUAlity slicing foR the Deployment of Security services.
ACROSS	Autonomous Control for a Reliable Internet of Services.
AG	Augmented Graph.
AGV	Automated Guided Vehiclex.
API	Application Programming Interface.
APP	Application Placement Problem.
AR	Augmented Reality.
ARPANET	Advanced Research Projects Agency Network.
AWS	Amazon Web Services.
BFS	Breadth First Search.
BS	Base Station.
BSS	Business Support System.
C-RAN	Cloud-Radio Access Network.
CAPP	Cloud Application Placement Problem.
CDN	Content Delivery Network.
CLF	CPU Load Factor.
COST	European Cooperation in Science & Technology.
CPU	Central Processing Unit.
CWI	Center for Mathematics and Computer Science.
DAG	Directed Acyclic Graph.
DB	database.
DFS	Depth First Search.
DHCP	Dynamic Host Control Protocol.
DNS	Domain Name Service.
DoS	Denial of Service.
DP	Dynamic Programming.

EC	Erasure Coding.
EEA	European Economic Area.
EM	Element Manager.
EMD	Elastic Media Distribution for online collaboration.
ETSI	European Telecommunications Standards Institute.
EU	European Union.
FaaS	Function as a Service.
FAN	Field Area Network.
FCAPS	fault, configuration, accounting, performance, security.
FEC	Forward Error Correction.
FUSE	Flexible federated Unified Service Environment.
FW	Firewall.
GA	Genetic Algorithm.
GCS	Greedy Chain Selection.
GDPR	General Data Protection Regulation.
GITA	Applied Telecommunication Research Group.
GRECO	Genetic Reliable ClOuds.
GRMP	Generalized Replica Management Problem.
GT-ITM	Georgia Tech Internetwork Topology Models.
HDFS	Hadoop Distributed File System.
HMI	human-machine interaction.
HPC	High-Performance Computing.
IaaS	Infrastructure as a Service.
IDS	Intrusion Detection System.
iFEST	improved Festival Experience with wearable Sensor Technology.
ILP	Integer Linear Program.
IMCF	Integer MCF.
InP	Infrastructure Provider.
IoS	Internet of Services.
IoT	Internet of Things.
IP	Internet Protocol.
ISP	Internet Service Provider.
IT	Information Technology.
KPI	key Performance Indicator.

LAN	Local Area Network.
LB	Load Balancer.
LC	Location Constraint.
LC-VNE	Location-Constrained VNE.
LP	Linear Program.
M2C	machine-to-cloud.
M2M	machine-to-machine.
MANO	Management and Orchestration.
MAV	Micro Air Vehicle.
MCF	Multi Commodity Flow.
MCTS	Monte Carlo Tree Search.
MEC	Multi-access Edge Computing.
MI	Management Interval.
MILP	Mixed Integer Linear Program.
MIP	Mixed Integer Program.
MIQCP	Mixed Integer Quadratically Constrained Program.
MKP	Multidimensional Knapsack Problem.
MPC	Model Predictive Control.
MPLS	Multi Protocol Label Switching.
MPTCP	Multi Path TCP.
MTDL	Mean Time to Dataloss.
MTTF	Mean Time To Failure.
NAT	Network Address Translation.
NCP	Network Control Program.
NF	Network Function.
NFV	Network Functions Virtualization.
NFVI	NFV Infrastructure.
NIC	Network Interface Card.
NS	Network Service.
OASIS	Organization for the Advancement of Structured Information Standards.
OS	Operating System.
OSI	Open Systems Interconnection.
OSS	Operations Support System.
P2P	Peer to Peer.
PaaS	Platform as a Service.
PC	Placement Configuration.

PL	Physical Link.
PM	Physical Machine.
PoP	Points of Presence.
PP	Physical Path.
PSO	Particle Swarm Optimization.
QAP	Quadratic Assignment Problem.
QoS	Quality of Service.
RAN	Radio Access Network.
RF	Radio Frequency.
RFID	Radio-Frequency IDentification.
RL	Replication Level.
RUV	Resource Usage Vector.
S3	Simple Storage Service.
SaaS	Software as a Service.
SAE	Service Architecture Evolution.
SDN	Software Defined Networking.
SECC	Service Embedding and Chain Composition.
SeP	Service Provider.
SFC	Service Function Chain.
SFT	Service Function Tree.
SG	Substrate Graph.
ShP	Shortest Path.
SL	Substrate Link.
SLA	Service Level Agreement.
SNe	Substrate Network.
SNo	Substrate Node.
SOA	Service Oriented Architecture.
SR	Service Requirement.
SVNE	Survivable Virtual Network Embedding.
TCP	Transmission Control Protocol.
TOCSA	Topology and Orchestration Specification for Cloud Applications.
UE	User Equipment.
USA	United States of America.
vCPE	virtual Customer Premises Equipment.
VIM	Virtualized Infrastructure Manager.

VL	Virtual Link.
VLAIO	Flanders Innovation & Entrepreneurship.
VLAN	Virtual Local Area Network.
VM	Virtual Machine.
VMP	Virtual Machine Placement.
VNE	Virtual Network Embedding.
VNe	Virtual Network.
VNF	Virtual Network Function.
VNF-FG	VNF-Forwarding Graph.
VNFM	VNF Manager.
VNFR	VNF Request.
VNo	Virtual Node.
VoD	Video on Demand.
VPC	Virtual Private Cloud.
VPN	Virtual Private Network.
VUPIC	Virtual Machine Usage Based Placement in IaaS Cloud.
WAN	Wide Area Network.
WSN	Wireless Sensor Network.

Chapter 1

Introduction

1.1 Research context

The Internet was conceptualized as an interconnection of networks. It has had a tremendous success and changed how we humans interact, spend our free time, consume information and do business. Over its lifetime, how the Internet is used, has evolved tremendously. One of the Internet's first killer applications was the worldwide web, allowing people to display a copy of a web page, that is stored on a remote server, on their personal computer. In this light, Al Gore, former vice president of the United States of America (USA), famously referred to the Internet as an information superhighway [1].

Traditionally, providers had to manage their own server infrastructure in order to deploy their Internet services. They had to decide on how many servers to buy and operate in order to deal with the expected demand for their service. While installing servers is a very slow process, taking up a few hours or even days, the demand for a service can vary rapidly in a matter of seconds. Hosting a service was complicated as the provider had to manage his own infrastructure and had to carefully provision resources upfront, requiring large capital investments. Overprovisioning resources, maximized service uptime, but lead to low infrastructure-utilization, while underprovisioning reduced wasted resources, but caused service downtime during demand peaks.

The introduction of cloud computing has removed this barrier of providing a service. The cloud computing paradigm offers users access to vast quantities of computing, storage and network resources on-demand. For instance, on-demand computing platform Amazon Web Services (AWS) was publicly launched in 2006. Here, the infrastructure was centralized in large datacenters and the resources could be time-shared, driving down the cost of computing drastically. Virtualization allowed multiple users to share infrastructure, improving resource utilization. Through replication of data and Virtual Machines (VMs) over multiple availability zones, combined with highly automated management software, the cloud enabled high availability services on top of commodity hardware. For instance, cloud storage service Dropbox [2], providing an online backup of personal data, was released in 2007. About the same time, broadband services delivering music and Video on Demand (VoD) were launched, e.g. Spotify [3] and YouTube [4], respectively. Compared to more traditional web applications,

these broadband services often were very demanding in terms of Quality of Service (QoS) and throughput. While the use of the Internet changed dramatically, the network protocols stayed the same. To deal with this ossification, network virtualization was proposed, enabling multiple logically separated overlay networks to coexist and employ their own network protocols, on top of shared networking infrastructure. In this way, network virtualization [5], combined with Software Defined Networking (SDN) [6] allowed assigning different QoS levels and bandwidth guarantees to traffic flows.

Cloud computing initially moved computing away from the end user, to remote centralized datacenters. Lately, Network Services (NSs) are increasingly being deployed across a geo-distributed cloud infrastructure. This geo-distribution is needed for several reasons. First, there are legal reasons and privacy concerns. For instance, the General Data Protection Regulation (GDPR) is a regulation in European Union (EU) law on data protection and privacy for all individuals within the EU and the European Economic Area (EEA). This regulation, implemented in 2016, requires that before personal data can be transferred outside the EU and EEA, to countries that are not deemed adequate, additional safeguards have to be put in place [7]. Concerns regarding where privacy-sensitive data is stored also crop up on a much smaller scale, within regional and national borders. For instance, in smart camera applications, privacy video material may not be allowed to leave the premises of a correctional facility or an oil platform. Second, while previously uptime was an application developer's main concern, now reaction times are becoming more important. Cloud elements that are in close proximity to the user's location generally can be accessed with a much lower latency compared to elements that are farther away. Consequently, resources that are hosted at the network edge can serve users much faster. For instance, a nearby web proxy can significantly reduce page loading times. Further, these nearby elements can result in significant bandwidth savings. For instance, in the smart camera scenario, the amount of video data generated by this application may be so overwhelming that not all data can be transferred to a remote datacenter due to bandwidth limitations. In this case, the data must initially be stored at the edge location closest to its source and only after a first analysis, a subset of the data is transferred to a remote datacenter.

In a first phase of geo-distribution, public and private clouds became interconnected, forming hybrid clouds. Netflix [8] is an architecturally interesting example of a global VoD streaming service. Its data and control plane are deployed on separate infrastructure. Anything that does not involve serving video is handled by AWS. In order to distribute its content efficiently, Netflix built a proprietary Content Delivery Network (CDN), i.e. OpenConnect, by partnering with local Internet Service Providers (ISPs). Interestingly, Spotify initially combined their own servers with a Peer to Peer (P2P) network, where music is cached at the client side and shared with other users. In 2011, 80% of the songs were delivered through the P2P network, reducing the buffering time for clients [9]. Initially, its master catalogue was hosted on Spotify's own datacenter. Later, it was migrated to Amazon Simple Storage Service (S3). In 2014, Spotify started phasing out its P2P network. All songs could now be streamed directly from the cloud, with acceptable delays. Finally, in 2016 Spotify migrated all of its computing resources to Google Cloud. One of the reasons for migrating to Google Cloud, was its Virtual Private Cloud (VPC) offering, which through network virtualization prevents the failure of one software component, to bring down the entire Spotify platform [10]. In contrast to Spotify

and YouTube, Twitch's crowd-sourced interactive live streaming platform [11] launched in 2011, does not provide the sources of live streaming [12]. Instead, it serves as a platform that interconnects sources and viewers, which complicates the orchestration of such a NS. First, the sources are no longer professional content providers and often have limited computation and network resources and may join or leave at any time, making high-quality live streaming more challenging. Moreover, Twitch allows interaction with live content broadcasters. The delay must therefore be limited to a few seconds. The main source of stream unavailability in Twitch is due to connectivity problems at the source side [12]. Moreover, there are challenges related to data persistence as, all video streams are archived and available as VoD during a period of 14 and 60 days for regular and premium broadcasters, respectively. Second, there are challenges related to the composition of the NS. Since sources and targets include computers, gaming consoles and mobile devices, premium broadcasters can opt for an online transcoding service in order to maximize their viewership. Further, users can customize the video composition, by making a selection, combining multiple viewing angles and live commentary. YouTube mainly offers static VoD and can therefore use a static caching strategy that heavily relies on placing CDN nodes close to the end user [13]. Driven by the delay sensitivity of live streaming, Twitch progressively and proactively replicates streams across servers only after sufficient demand is observed. This more centralized management approach places a much greater reliance on effective peering and interconnection strategies. At the same time, users must be able to switch between sources with a delay of at most a few seconds.

In a second phase of geo-distribution, cloud elements other than CDN nodes, are being placed at the edge of the network, further complicating the cloud management. According to a recent survey on edge computing for the Internet of Things (IoT) [14], this edge infrastructure is needed to enable ultra-low response times and facilitate data processing for future IoT applications. While the compute and storage capabilities of centralized cloud elements are immense, the capabilities of IoT devices are mostly limited to messaging. Cloud robotics and self-driving cars will offload the intelligence required to make realtime decisions to the cloud. The required response times will be in the order of milliseconds, impeding offloading computation to a remote centralized cloud. The devices at the edge, will have capabilities that are more limited than the ones in centralized datacenters, but that are accessible at lower latencies. In order to enable ultra-low response times for these demanding IoT applications, mobile providers are investigating placement of Multi-access Edge Computing (MEC) sites in their Radio Access Network (RAN). In the next generation of mobile networks, i.e. 5G, this MEC can be used by the mobile operator to place Virtual Network Functions (VNFs) close to the user and can be opened up to third party Service Providers (SePs). The architecture of the geo-distributed cloud infrastructure is shown in Figure 1.1. The core network interconnects the centralized clouds and the edge clouds. The Local Area Network (LAN) interconnects the end-devices and connects the end-devices to the edge clouds. While edge clouds are closer to the end-user, the capability and reliability of this infrastructure is much more limited, compared to centralized clouds. Further, the infrastructure in centralized clouds is much more homogeneous, with heterogeneity being limited to multiple generations of servers being used.

According to Yu et al. [14], the long-term analytics for big data applications will be performed in centralized clouds, while short-term storage and analytics will be performed

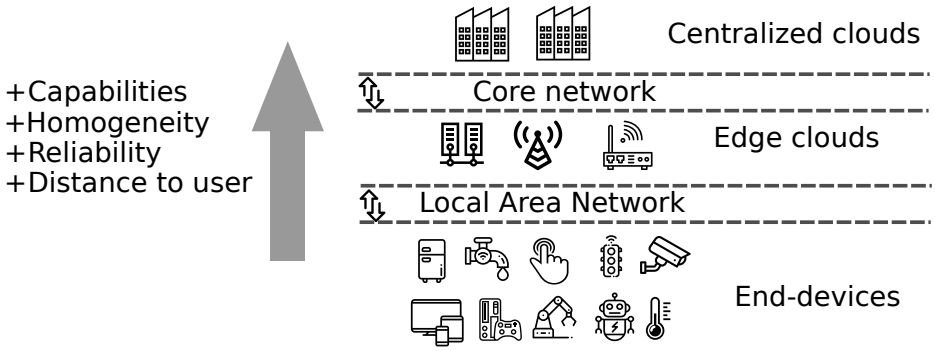


Figure 1.1: The architecture of geo-distributed cloud infrastructure.

at the edge. Since the devices at the edge are severely resource-constrained, intelligent orchestration algorithms are needed that can deal with these heterogeneous resources. These algorithms must orchestrate the services, while at the same time considering the desired outcome of the service and the scarcity of the compute, network and storage resources.

1.2 Problem statement

In the previous section, we described the growing need to deploy Internet services across a geo-graphically distributed cloud infrastructure. This need stems from a combination of legal and privacy concerns; the growing demand for responsive services with low reaction times; and the need to reduce the uplink bandwidth requirements of bandwidth-hungry applications towards the core network. In response, SePs are increasingly interconnecting cloud elements across multiple datacenters and cloud elements are being placed at the edge of the network.

In order to operate cloud environments, highly-automated cloud management software is needed. Novel management algorithms are needed as the state of the art cannot deal with the increased heterogeneity caused by introducing infrastructure at the edge into the cloud environment. The aim of this thesis is to investigate the challenges related to the orchestration of services across these heterogeneous cloud environments and propose novel management approaches that address these challenges. The research questions that are investigated in this thesis are presented in Section 1.3.

1.3 Research questions

In this thesis, the following research questions and subquestions regarding the management of services in heterogeneous cloud environments will be investigated. First, there are queries regarding the replication of persistent application data. While the storage capabilities within datacenters are virtually infinite, the capabilities at the edge of the network are a great deal scarcer. Moreover, the storage capabilities differ wildly from one edge location to another. While in one edge location, only a single gateway with some storage capabilities might be present, at other edge locations the data can possibly be replicated across multiple storage

racks. The bandwidth capabilities both within and across storage cells vary wildly from one edge location to another. The storage locations can be interconnected through a plethora of communication technologies, either via high-bandwidth fiber-optical cables, copper wires or even via the wireless Radio Frequency (RF) spectrum. Further, data from a wide range of services will coexist at the edge. These services have diverse QoS requirements regarding data persistence, i.e. how long the data must be stored and to which degree data loss can be tolerated. The services will compete for the same physical resources. Thus, for geo-distributed clouds to be operated effectively, the following questions regarding data management at the edge must be answered.

Question I. How to effectively replicate data across storage nodes in heterogeneous cloud environments?

Subquestion Ia. How to maximize the expected provider revenue over a service's lifetime?

Subquestion Ib. How to model data loss in replicated systems?

Subquestion Ic. What is the impact of Service Level Agreements (SLAs) on data availability on the optimal replication strategy?

Subquestion Id. What is the impact of the cloud characteristics on the optimal replication strategy?

Subquestion Ie. How does a replication strategy that balances SLA conformity with storage costs perform, compared to traditional replication strategies?

Second, there are queries regarding the provisioning of reliable Survivable Virtual Network Embeddings (SVNEs). Traditional cloud management often ignores bandwidth limitations within a datacenter, as the operator has full control over the infrastructure and the wired links can be heavily overprovisioned. However, when NSs are deployed in geo-distributed cloud environments, the network limitations cannot be ignored. Hence, in order to reliably orchestrate NSs, across geo-distributed cloud infrastructure, both nodal and bandwidth resources must be provisioned. Further, in order to protect realtime broadband services against failures in the underlying infrastructure, they must be able to survive failures in both nodes and their interconnections. Current protection schemes, introduce the same level of redundancy across the entire infrastructure, regardless of the local failure behavior and resource scarcity. Moreover, they typically introduce a predetermined level of replication for each service, regardless of the required QoS. Thus, to maximize the number of applications that can co-exist in these geo-distributed cloud environments, while satisfying their reliability requirements, the following questions must be answered.

Question II. How to protect NSs against node and link failure in heterogeneous cloud environments?

Subquestion IIa. How to deal with the scarcity of resources at the edge?

Subquestion IIb. How to introduce redundancy into the Placement Configuration (PC) only where it is needed?

Subquestion IIc. How to place applications, while guaranteeing a minimum availability for each application?

Subquestion IId. How to calculate the expected availability of a SVNE?

Subquestion IIe. How does a proactive approach that introduces a variable level of protection and that is availability-aware perform, compared to traditional strategies?

Third, there are queries regarding the orchestration of NSs in NFV environments. NSs are increasingly being described as a composition of VNF instances and their required interconnecting Virtual Link (VL) instances. To successfully orchestrate a NS in an NFV environment, its VNF-FG must be composed based on the Service Requirements (SRs) and the composed VNF-FG must be embedded onto the infrastructure. The orchestration of these services across a geo-distributed cloud infrastructure is very challenging as compared to traditional Information Technology (IT) services, because their execution often depends on very specific hardware capabilities. Further, these NSs originate and terminate at specific geographical locations, whose interconnections introduce severe bandwidth limitations. The heterogeneity in the SNe is known to complicate the service-embedding significantly. Further, often multiple candidate VNF-FG compositions can fulfill the SRs, drastically increasing the search space for PCs. Traditional orchestration approaches consider the composition and embedding problems in two separate stages. Composing the service without any knowledge about the SNe can result in a VNF-FG that is hard or even impossible to embed. Hence, coordination between these two tasks is expected to improve the quality of the PC. Further, current service composition models make very restrictive assumptions on the service topology and requirements, which severely limit their applicability. Thus, to effectively orchestrate services in geo-distributed NFV environments, the following questions must be answered.

Question III. How to orchestrate NSs in NFV environments?

Subquestion IIIa. How to compose the VNFs based on the service requirements?

Subquestion IIIb. How to coordinate the VNF-FG composition and embedding?

Subquestion IIIc. How do orchestration approaches that coordinate the composition and embedding tasks perform, compared to uncoordinated approaches?

Subquestion IIId. How to orchestrate NSs that require traffic aggregation?

1.4 Contributions

The research work presented in this monograph has been internationally validated in different networking peer reviewed journals and conferences. Several experts have provided their comments in the peer reviews allowing us to improve our investigations and to guide the direction of our research. The articles published during the development of this thesis are detailed in Table 1.1 and 1.2 with a quality indicator that highlight the quality level of the journal/conference.

Table 1.1: Publications in journals.

Year	Paper title	Journal	Quality indicator
2017	Resilient application placement for geo-distributed cloud networks [C1]	Journal of Network and Computer Applications	Impact Factor 3.991, Q_1
2018	Coordinated service composition and embedding of 5G location-constrained network functions [C2]	IEEE Transactions on Network and Service Management	Impact Factor 3.286, Q_1
2019	Delay-constrained NFV Orchestration for Heterogeneous Cloud Networks [C3]	IEEE/ACM Transactions on Networking	Impact Factor 3.11, Q_1 status: submitted

Table 1.2: International conferences.

Year	Paper title	Conference	Quality indicator
2014	Towards a fluid cloud: an extension of the cloud into the local network [C4]	International Conference on Autonomous Infrastructure, Management and Security (AIMS), PhD track	-
2015	Fault-tolerant application placement in heterogeneous cloud environments [C5]	Network and Service Management (CNSM)	CORE Ranking B, acceptance rate 17.6%
2017	Cost-effective replica management in fault-tolerant cloud environments [C6]	Network and Service Management (CNSM)	CORE Ranking B, acceptance rate 17.6%
2019	Effective NFV Orchestration for Wide-Ranging Services Across Heterogeneous Cloud Networks [C7]	Symposium on Integrated Network and Service Management (IM)	CORE Ranking A (2017)

1.5 Organization

The organization of this thesis is shown in Figure 1.2. There, the individual chapters and their interdependencies are shown. In Chapter 2, the challenges related to resilience in cloud computing, VNE and NFV composition are identified.

The contributions of this thesis are presented in Chapters 3, 4, 5 and 6. Chapter 3 focuses on Question I. It addresses the challenges related to balancing resilience and operational costs in dynamic cloud environments. It defines the Generalized Replica Management Problem (GRMP) which considers the problem of maximizing the provider revenue for a replicated

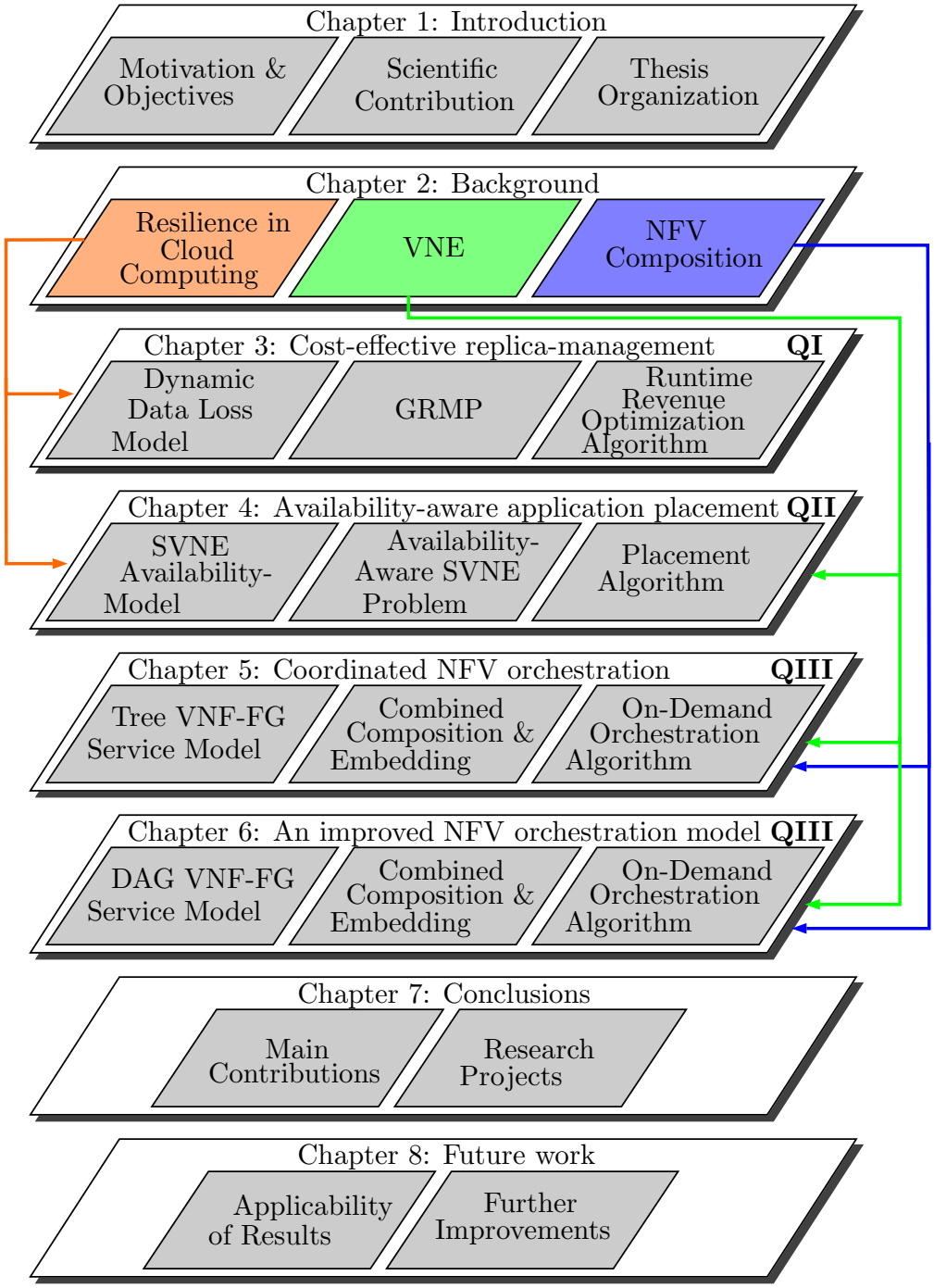


Figure 1.2: Thesis organization.

service over the service lifetime. This problem is the task to periodically monitor the service's health and decide whether scaling in or out maximizes the provider's monetary reward, while consider the SLA. Instead of directly considering the status of each individual PL and PM, our proposed approach considers the impact of failure distribution, operational costs, recovery times and scheme. A Dynamic Programming (DP) is proposed that performs a runtime revenue optimization over the service's lifetime. This algorithm's execution time depends on the used replication scheme, but not directly on the dimensions of the SNe, which makes the approach well-suited for the management of replicated services in large-scale environments.

Chapter 4 focuses on Question II. It focuses on the challenges related to the protection of VNEs in heterogeneous cloud environments. The chapter proposes a fine-grained replication scheme based on the embedding of multiple service duplicates and develops the corresponding service availability model. Compared to traditional replication schemes, these duplicates can share resources. Building on this availability model, the availability-aware SVNE problem is formulated, which aims to embed a service with minimal cost on a SNe, where both PMs and PLs can fail, while guaranteeing a minimum availability per service. Since finding an exact solution is NP-hard, several placement heuristics are proposed. As the uplink capabilities of IoT environments are often unreliable or nonexistent, the cloud management algorithms might need to be executed on these unreliable devices on the edge. However, the devices at the edge often have very limited computational capabilities and increasingly fail without any warning, compared to the servers in centralized datacenters. Therefore, a distributed fault-tolerant placement algorithm is proposed that can find a solution to the availability-aware SVNE problem. The proposed Genetic Algorithm (GA) uses a distributed pool model to distribute the solution population over the workers. The workers, executed on the edge devices each retrieve a set of individuals from the database, process them and write them back. When a worker fails during processing, its operation times out and another worker can start processing these individuals. Finally, a scalable centralized heuristic, based on subgraph isomorphism is proposed, that can find solutions in a SNe with up to 100 PMs under 20 seconds.

Chapters 5 and 6 focus on Question III. They both consider the problem of service orchestration in NFV environments. Service composition in NFV environments entails both service composition and VNF-FG embedding. The composition of the NS's VNF-FG is based on the SRs. VNF-FG embedding is closely related to the VNE problem. Two service models are considers. First, Chapter 5 considers this problem, building on an existing service model can compose VNF-FGs with a tree topology. The problem is formulated as an Integer Linear Program (ILP) that can be used to find the exact solution for smaller problem instances. To deal with the computational complexity of the problem, a fast, greedy heuristic that iteratively adds SFCs to the composition and embeds them at the same time, is proposed. Second, Chapter 5 proposes an improved service model that provides wider applicability. It supports traffic aggregation, bidirectional chaining requirements and optional VLs. This combined composition and embedding problem is formulated as an ILP that can be used to find the exact solution for smaller problem instances. Further, a recursive heuristic is proposed that can orchestrate services that adhere to this service model.

Finally, Chapter 7 presents the conclusions and main results of this thesis. Chapter 8 depicts the possible improvements of the presented contributions and the emerging research branches that can be the subject of future work.

Chapter 2

Background

This chapter provides the basic background against which the thesis is developed. First, the cloud computing paradigm and the recent trend towards geo-distribution of the cloud infrastructure are introduced. Second, the concept of network virtualization is introduced. Third, NFV, i.e. the process to virtualize Network Functions (NFs) that previously ran on dedicated hardware, is discussed. Then, the resource allocation challenges related to the management of these computing environments are presented.

2.1 Cloud computing

Cloud computing is a way of offering computing, storage and networking resources, on-demand, over the Internet. Previously, IT service providers had to manage their own private servers, requiring large up-front investments. Nowadays, cloud computing provides access to a vast pool of computing resources, realized by commodity hardware that resides within large datacenters. The tenants time-share this infrastructure, and must not be aware of each other's presence. The required performance isolation is realized through combinations of virtualization techniques and highly automated cloud management software. Due to the centralization of computing resources, the cloud formalism promises reduced costs due to the economies of scale. Further, it offers the possibility to scale an application in or out, when the demand for an application in- or decreases, respectively. Hence, the service provider no longer has to heavily over-provision resources. Further, the cloud offers highly available services running on top of commodity hardware, enabled by virtualization techniques. For more information on the cloud computing paradigm, the interested reader is referred to [15, 16].

2.1.1 Cloud provisioning models

The cloud paradigm liberates service providers from the burden to manage the entire stack required to realize their service, from the physical infrastructure, up to managing their application. This burden is now shared by a set of stakeholders. Jennings and Stadler identify three cloud actors in the cloud model [16].

- *Cloud provider*: manages a set of datacenter hardware and system software resources, providing abstractions of those resources. The cloud provider is responsible for allocating these resources so that it meets SLAs and/or achieves other management goals;
- *Cloud user*: uses public clouds to host applications that it offers to its end users. The cloud user is responsible for meeting SLAs agreed with its customers (i.e., end users) and is typically concerned with doing so in a manner that minimizes its costs and maximizes its profits by ensuring that the level of resources leased from the cloud provider scales in line with demands from its end users;
- *End user*: generates the workloads that are processed using cloud resources.

Cloud resources are offered in one of the following resource models. These models offer the computation resource using different levels of abstraction, each characterized by its own granularity of control and separation of concern.

- *Software as a Service (SaaS)*: does not consider a separate cloud user. The cloud provider manages the infrastructure, the platform and the application. The cloud provider offers the service directly to the end user. Examples of SaaS include Google Apps [17], Dropbox [2] and GoToMeeting [18].
- *Platform as a Service (PaaS)*: provides a platform for software creation. It allows businesses to design and create applications that are built into the PaaS with special software components. In PaaS, the cloud provider manages the infrastructure; the cloud user manages the platform and the cloud user manages the application.
- *Infrastructure as a Service (IaaS)*: provides the tenants access to a virtualized datacenter. As opposed to SaaS or PaaS, IaaS clients are responsible for managing aspects such as applications, runtime, Operating Systems (OSs) and data. Providers of the IaaS cloud manage the servers, hard drives, networking virtualization and storage. Examples of IaaS include AWS [19], Rackspace [20] and Google Compute Engine [21].
- *Function as a Service (FaaS)*: provides a platform that executes functions in response to events at any scale. The user is only billed for the time during which the functions are invoked. The functions are short-lived, typically the execution time of a function is limited to a few minutes. The state of the application is stored in a distributed database; the functions are stateless. The key difference compared to PaaS, is that during the lifetime of a PaaS application at least one of the servers must be online, while that is not required for FaaS. Examples of FaaS platforms include OpenWhisk [22], OpenLambda [23] and Azure Functions [24].

2.1.2 Geo-distribution

While traditionally a cloud infrastructure is located within a single datacenter, recently, there is a need for geographical distribution. NSs now often span multiple federated clouds. Geo-decentralization is needed to keep up with the users' ever-increasing demand for low response times and high throughput. There are also legal reasons, restricting the storage or processing

of privacy-sensitive data to particular geographical locations, e.g., within the boundaries of a country.

Lately, this need for geo-distribution has led to a new evolution of decentralization. Most notably, the extension of cloud computing towards the edge of the network, is generally referred to as fog or edge computing [25]. In fog computing, computation is performed at the edge of the network at the gateway devices, reducing bandwidth requirements, latency, and the need for communicating data to the servers. These gateway devices will aggregate the data originating from a wide variety of things or objects, e.g. Radio-Frequency IDentification (RFID) tags, sensors, actuators and mobile phones, in what is known as IoT [26]. Second, mist computing pushes processing even further to the network-edge, involving the sensor and actuator devices [27]. Closely related to mist computing is the cloud robotics architecture put forward by Hu et al. [28]. The architecture leverages the combination of a virtual ad-hoc cloud formed by machine-to-machine (M2M) communications among participating robots, and an infrastructure-cloud enabled by machine-to-cloud (M2C) communications.

2.2 Virtual networking

The cloud paradigm offers computational resources on-demand to its tenants. While the users time-share the infrastructure, this should be transparent to its tenants. First, the tenants require a simpler, abstract view on the resources, where the configurations of the different tenants do not affect one another. Second, the resource management by the provider should isolate the performance of multiple tenants. To realize this, the cloud formalism relies on a combination of node and link virtualization.

2.2.1 Node virtualization

The processes required by the service are executed in virtualized environments, referred to as Virtual Nodes (VNos). These virtualized environments are deployed on PMs that run a host OS. The two dominant virtualization environments are VMs and containers. These environments differ in how much overhead they require. On the one hand, each VM image contains a guest OS that manages the user processes. While running a host OS and multiple guest OSs at the same time causes a significant overhead, it does bring a great deal of flexibility since VMs executed on the same PM can run different OSs, e.g., Ubuntu 18.04, 16.04 and Windows 10 at the same time. Communication between processes running within the same VM is managed by the guest OS. Performance-isolation between VMs collocated on the same machine, is typically realized by assigning them to different Central Processing Unit (CPU). This process is referred to as CPU pinning. On the other hand, the processes in a containerized environment are all directly executed within the host OS. Processes can only observe and talk to processes that correspond to the appropriate namespace. Typically, the relative performance of the containers that run on the same PM, is configured through their assigned priorities. The resource requirements of VNos are typically expressed along the following dimensions.

- *Processing*: a measure for the CPU requirements of a VNo. When the PMs in the SNe are heterogeneous, then the processing requirements to execute this VNo can differ from

PM to PM. Typically, these requirements are expressed in the number of cores, or by a scalar proportional to the number of instructions that must be processed by the VNo to assure acceptable behavior;

- *Memory*: the required space in the working memory to host the VNo. It is typically expressed in MB or GB.

- *Storage*: the amount of storage required on persistent memory. It is typically expressed in MB or GB.

- *Network Interface Card (NIC) bandwidth*: the overall bandwidth requirement of the VNo.

The lifecycle of a VNo is the following. Before an instance of a binary image can be *deployed* on a PM, its content is transferred over the network. When an image is deployed, its processes are loaded in the PM's memory. Then, images can be *suspended*, meaning that its processes are halted, after which the image can be either *destroyed* or *resumed*. Further, an image can be *migrated* to another PM. Migration is a processing- and bandwidth-intensive operation and typically always results in some minimal service downtime. When an image is suspended, prior to migration, the migration process is referred to as cold migration. When the image remains operational during transfer, the process is referred to as live migration.

2.2.2 Link virtualization

The communication between processes is organized using VLs. For security reasons, it is important that the traffic from different tenants is logically separated. Logical separation of traffic flows can be realized through Virtual Local Area Network (VLAN) technology or via the creation of Virtual Private Network (VPN) tunnels. These tunnels create logical end-to-end connections, referred to as VL instances that are routed over Physical Paths (PPs). A PP is formed by a sequence of PLs in the SNe. The routing of a VL should be transparent to the user. Further, the traffic flows can use different network protocols, optimized for the tenant's requirements. The data flowing over these VLs is typically encrypted to prevent eavesdropping.

Performance isolation between users sharing network resources can be realized via prioritization of traffic flows using Multi Protocol Label Switching (MPLS), or through SDN. Traditional switches use endpoint routing, meaning that the interface to which a packet is forwarded by a router depends solely on the destination IP-address of the packet. In contrast, SDN separates the control and data plane, facilitating the management of this infrastructure, by enabling centralized steering of the traffic flows through the network. SDN enables the infrastructure provider to provide bandwidth guarantees to different flows.

The requirements for VLs are typically expressed along the following dimensions.

- *Bandwidth*: the required communication bandwidth between the end-points of the VL.

- *Delay*: maximum acceptable delay, e.g., expressed in ms or hops, between the end-points of the VL.

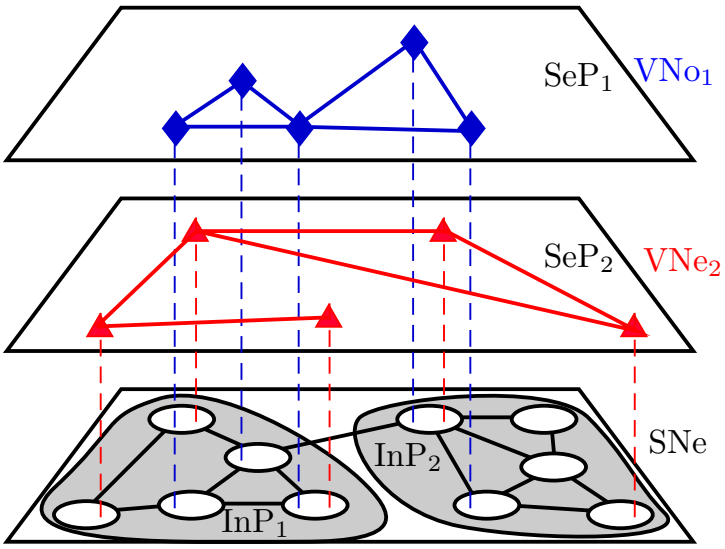


Figure 2.1: Network virtualization environment.

2.2.3 Network virtualization

Network virtualization is the process of combining hardware and software network resources and network functionality into a single software-based administrative entity. The result of this process is a Virtual Network (VNe).

Feamster et al. propose a business model similar to IaaS for network virtualization [29]. In this model, there are two actors. First, the Infrastructure Provider (InP) provides the infrastructure. Second, the SeP provides the NS.

Figure 2.1 illustrates the network virtualization environment, formed by two SePs, i.e., SeP_1 and SeP_2; and two InPs, i.e., InP_1 and InP_2. In this illustration, SeP_1 and SeP_2 each request a single VNe, i.e. VNe_1 and VNe_2, respectively. The VNes form an overlay-network, comprising VNo instances, interconnected by VLs.

In the context of network virtualization, a VNo is typically a software component with routing functionality. Hence, it also referred to as a virtual router. A VL (instance) is a logical interconnection of two virtual routers, appearing to them as a direct PL with dynamically changing properties. The virtualized resources in these VNes are realized by resources in the SNe. Several independent VNes can coexist at the same time on top of a SNe. The SNe, formed by pooling resources from InP_1 and InP_2, comprises Substrate Nodes (SNos), interconnected by Substrate Links (SLs). These resources can either be physical network resources, i.e., PLs and PMs, or they can be virtualized resources too, possibly leased from another InP.

2.3 Network functions virtualization

Similar to how cloud computing revolutionized how traditional IT services are deployed, NFV is doing the same for broadband services that require NFs, e.g., Network Address Translation

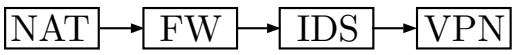


Figure 2.2: Illustration of a VNF-FG, comprising 4 VNF instances and 3 VL instances.

(NAT), Firewall (FW), Intrusion Detection System (IDS) and VPN, that previously ran on middle-boxes. Section 2.3.1 discusses why these middle-boxes were installed in the network. NFV allows spinning up and down these NFs, with the same flexibility as if they were VMs in an IaaS cloud. Compared to network virtualization (c.f. Section 2.2.3), NFV considers VNos that realize more complex NFs, than only forwarding. Section 2.3.2 provides an overview of the most common VNFs, most of which were previously executed on middle-boxes.

In NFV environments, the logical structure of a NS is represented by its VNF-FG. This VNF-FG is essentially a VNe, where the VNos are VNF instances. An illustration of a VNF-FG is shown in Figure 2.2.

This VNF-FG comprises NAT, FW, IDS and VPN instances. In order to promote re-usability and modularity, NSs that are deployed in NFV environments are typically described as a composition of VNFs. Each VNF is described by a VNF template, detailing its nodal and connectivity requirements. For instance, Topology and Orchestration Specification for Cloud Applications (TOCSA) [30] is a data model standard by industry group Organization for the Advancement of Structured Information Standards (OASIS) that can be used to orchestrate NFV services and applications. Most importantly, this standard describes VNF types, containing the constraints in terms of architecture, processing and memory requirements to host an instance of this VNF on a SNo and the total maximum and minimum number of instances of this VNF. Further, it describes relationships between VNFs, i.e., which VNF is contained within another and which connections it can have to other VNFs. The resource requirements of VNF instances are typically dependent on the applied workload. It is important to distinguish between the templates that describe the virtualized resources and the deployments, adhering to these templates. In this monograph, the templates that describe the VNos are referred to as VNFs. The actual deployments of these templates are referred to as VNF instances. While the resource requirements of VNFs are relative, the requirements of their instances in the VNF-FGs are absolute. The possible connections between VNFs, as specified in their templates, are referred to as VLs. For an ingress VL, only the target VNF is fixed, while for an egress VL, only the source VNF is predetermined. VL instances are the connections between VNF instances in the VNF-FG, formed by connecting an egress VL of one VNF instance to an ingress VL of another VNF instance. While the requirements of a VL depend on the workload, the requirements of a VL instance are known.

2.3.1 The need for middle-boxes

Middle-boxes are dedicated, proprietary hardware, that fulfill a particular NF. Historically, these boxes were installed by infrastructure providers, to deal in an ad-hoc way with the new issues that they were facing, while the architecture of the Internet architecture and protocols stayed largely the same. Conceptually, the Internet was designed as dumb packet forwarder, with the intelligence located at the edge of the network. It evolved as an experimental packet-switched network. The Open Systems Interconnection (OSI) data communications reference

model divides the communication functions of a telecommunication system over seven layers. In this model, the packets should only be processed in the network stack up to the network layer, i.e. layer three. The transport, session, presentation and application logic should only be processed in the end-devices, i.e. the hosts. The first Internet applications were simple web and e-mail services. In the past two decades, Internet services were deployed that had not been envisaged upon the conception of the Internet, including music and video streaming services, e.g., Spotify and Netflix, massive online multiplayer games and social media, e.g., Facebook and LinkedIn. However, the basic Internet Protocols (IPs) have not changed since 1983, when Advanced Research Projects Agency Network (ARPANET) changed from Network Control Program (NCP) to the Transmission Control Protocol (TCP)/IP protocol suite, since it would be very hard to remain backwards compatibility with older devices. This inertia is often referred to as the *ossification* of the Internet. In order to support these novel broadband services, the service providers have installed a wide variety of middle-boxes into the network.

2.3.2 VNF taxonomy

Table 2.1: VNF taxonomy.

Name	Layer	Type	Description	Flexibility	Ref.
Storage VNF	Application	Functional	Replica of data.	Optional	[12]
Authentication Authorization Accounting (AAA) (e.g. RADIUS)	Application	Functional	Controls access to data.	Sequence	[31]
Domain Name Service (DNS)	Application	Functional	Resolve host names.	Optional	[32]
Route Reflectors	Application	Scalability	An alternative to the logical full-mesh requirement of IBGP.	Optional	
Dynamic Host Control Protocol (DHCP)	Application	Functional	Assigns IP-addresses.	Optional, sequence	
Service Architecture Evolution (SAE) gateway	Application	Performance	Change routing path according to application.	Optional, sequence	
Deep Packet Inspection (DPI)	Application	Security	Inspection of packets.	Optional, sequence	[33]
Load Balancer (LB)	Application	Performance	Divide the workload over multiple homogeneous servers, or separate flows based on their processing needs (e.g. video stream and web traffic).	Optional, sequence	[32]
Monitoring, Monitoring and Analytics probes	Application	Reliability	Monitor QoS, learn possible optimizations, troubleshoot traffic flows.	Optional, sequence	[33]
Lawful Interception	Application	Security	Wiretapping by law enforcement.	Optional, sequence	[33]
Watermarking	Application	Security	Identify ownership of the copyright, tracing copyright infringements, source tracking.	Optional, sequence	

Table 2.1: VNF taxonomy.

Name	Layer	Type	Description	Flexibility	Ref.
Transcoding (video or audio)	Application	Functional, performance	Change codec of a video, done because target device does not support source codec, or to reduce the bandwidth.	Optional, sequence	[12, 34]
Conferencing, recording, speech recognition, text to speech, language translation, event notification	Application	Functional	VNFs required for in teleconferencing or customer support.	Sequence	[31]
Performance enhancing proxy (PR)	Application	Performance	Limits retransmission of the same data.	Sequence, optional	[32, 31]
Cache (content distribution network)	Application	Performance	Limits retransmission of the same data.	Sequence, optional	[35, 31]
Virtual Customer Premises Equipment (vCPE)	Application	Functional	Parental control, VoD, Billing	Sequence, optional	[35, 31]
Retransmission engine	Application	Performance, reliability	Reduce latency in lossy networks.	Optional, sequence	[35]
Intrusion Detection System (IDS)	Application	Security	Detect anomalies.	Optional, sequence	[35, 31]
Virtual Private Network (VPN) tunnel	Application	Security	Avoid man in the middle attacks in non-trusted parts of the internet.	Optional, sequence	[35]
Anonymization VNF	Application	Security	Used when personal data cannot leave the enterprise network, without being anonymized first.	Optional, sequence	
Malware detection	Application	Security	Detect malware.	Optional, sequence	[35]
Billing	Application	Functional	Required for orchestration across multiple administration domains.	Optional, sequence	
Monitoring	Application	Reliability	Monitoring of both network and application	Optional, sequence	[35]
Streamer	Application	Functional	Creates a video or audio stream from source material.	Optional, sequence	
Composer	Application	Functional	Make a composition of multiple streams.	Optional, sequence	[34]
Cloud interfacer	Application	Functional	Allow different Cloud providers to work together.	Optional, sequence	
MAC VNFs	Data	Performance	QoS differentiation, WAN optimizers, low latency QoS schedulers.	Optional, sequence	
Multi Protocol Label Switching (MPLS)	Data, network	Performance	Separate flows. Label packets on layer 2 based on destination IP. Switching on layer 2 is less resource intensive.	Optional, sequence	[35]
FEC encoder and decoder	Data, Network, Transport	Performance, reliability	Reduce latency in lossy networks.	Optional, sequence	
Router (customer edge, provider edge)	Network	Functional	Forwards data packets.	Optional, sequence	[32]

Table 2.1: VNF taxonomy.

Name	Layer	Type	Description	Flexibility	Ref.
Header modification	Network	Functional	E.g., to change time-to-live of a packet.	Optional	
Slicing VNFs	Network	Performance	Provide differentiated QoS.	Optional	
Firewall (FW)	Network, Transport	Security	Monitors and control incoming and outgoing network traffic based on predetermined security rules.	Optional, sequence	[32]
Network Address Translation (NAT)	Network, Transport	Functional	Provide connectivity between multiple IP-ranges.	Optional	[32, 31]
Signal processing (e.g. Fast Fourier Transform)	Physical	Functional	Signal processing required for wireless communications.		[36]
Multipath TCP (MPTCP)	Transport	Performance	Split the traffic over multiple PPs in the SNe.	Optional, sequence	
Quic UDP Internet Connections (Quic)	Transport	Performance	Provides reduced connection and transport latency.	Optional	
Transport relays	Transport	Performance	Terminate the connection at an intermediate point. Useful when the latency of the infrastructure varies (e.g., latency wireless, and cloud).	Optional	
Session control	Transport	Functional	Management of communication sessions.	Sequence, optional	[35]
NAT-P	Transport, network	Functional	Provide connectivity spanning multiple protocols.	Optional, sequence	[32]
Rate control	Transport, session	Performance, security	Data rate limitation to enforce SLAs. Can prevent DoS attack.		

A comprehensive list of VNFs is given in Table 2.1, together with their characteristics. Many of these NFs were originally executed on middle-boxes and therefore appeared in the middle-box taxonomy by Carpenter et al. [32]. 'Layer' indicates up to which layer in the OSI data communications reference model the VNF processes the packets.

Three types of VNFs are used. *Functional* indicates that the VNF determines which service the NS realizes. *Security* and *performance* are non-functional aspects. For instance, an application proxy caches data, in order to avoid retrieving this data multiple times from the original source. Load Balancers (LBs) forward the traffic over a set of servers, in order to achieve a larger overall capacity. FWs block or pass packets, based on a set of rules regarding the source and destination IP-address and port of the packet. A NAT allows devices that have a local IP-address to connect to the Internet.

The category 'Flexibility' indicates the flexibility in the composition of the VNF-FG. Optional VNFs can be added to the NS, but in some cases, it is possible to compose a valid VNF-FG without this VNF. For instance, a VPN tunnel might only be required if the VL is routed over infrastructure that is not controlled by the cloud user. This scenario occurs in a hybrid cloud, where the infrastructure pools resource from a private cloud, operated by the cloud user and resources from a public cloud. Similarly, watermarking of a video stream might only be required if the stream leaves the enterprise network. Additionally, a start and end-point for a Multi Path TCP (MPTCP) connection might only be used if the SNe does not support

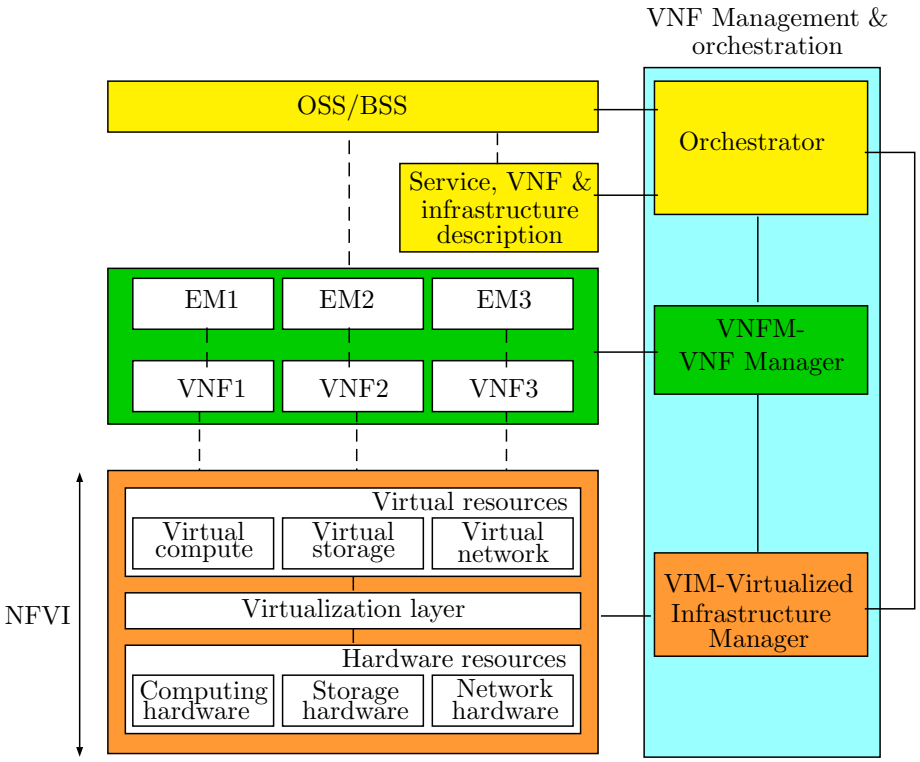


Figure 2.3: MANO architecture proposed by ETSI.

path splitting, or if there is not enough bandwidth along a specific PL. Header modification to increase the time-to-live of a packet is only required if the PP over which the packets are routed exceeds a maximum length. Further, there are VNFs of which their relative order in the chain does not affect the delivered functionality. For instance, the relative order of NAT and FW instances in the SFC shown in Figure 2.2, does not necessarily influence the functionality of this chain. However, changing the relative order of the IDS and VPN instance in the chain, must not be allowed. Encrypting the traffic in the start-point of the tunnel would hinder detection of intrusions later on.

2.3.3 Management and orchestration (MANO)

The NFV management and orchestration architecture proposed by European Telecommunications Standards Institute (ETSI) [37] is shown in Figure 2.3.

The following entities can be identified.

- *Element Manager (EM)*: is responsible for the functional management of VNF i.e. fault, configuration, accounting, performance, security (FCAPS). An EM can manage one or more VNFs.
- *VNF Manager (VNFM)*: performs the life-cycle management for one or more VNFs. The

VNF life-cycle is highly similar to that of a VNo in a network virtualization environment. VNF life-cycle management entails the following functions. VNF *onboarding* is the processing a VNF template, such that instances of it can be readily generated. VNF *deployment* and *undeployment* are the processes of creating and terminating a running VNF instance, respectively. *Monitoring* entails observing the key performance metrics of a running VNF. *Scaling up* or *down* is the process of in- or decreasing the dimensions of the VNF instance, e.g. by changing the number of assigned CPU cores or the amount of reserved working memory. *Healing* restores a deployed VNF in case of software- or hardware-related problems. In response, the current instance can either be migrated to another SNo, or a new instance can be created.

- *NFV Infrastructure (NFVI)*: the environment in which VNFs run. This includes physical resources, virtual resources and virtualization layer. Physical resources include computing, storage and networking infrastructure. Virtual resources are instantiated on these physical resources. The virtual resources are ultimately utilized by VNFs. The virtualization layer is responsible for abstracting physical resources into virtual resources.

- *Virtualized Infrastructure Manager (VIM)*: the VIM is responsible for controlling and managing the NFVI computing, networking and storage resources within one operator's infrastructure domain. It is also responsible for collection of performance measurements and events.

- *Orchestrator*: generates, maintains and tears down NSs that are composed of multiple VNF instances. The orchestrator enables the creation of end-to-end services that span infrastructure from multiple NFVIs. The resource allocation algorithms required to realize the orchestrator function will be the focus of this thesis. The resource allocation challenges related to the orchestration of NSs in NFV environments will be discussed in Section 2.4.

- *Operations Support System (OSS)/Business Support System (BSS)*: OSS deals with network management, fault management, configuration management and service management. BSS deals with customer management, product management and order management.

2.4 Resource allocation challenges

The structure of this section is the following. First, Section 2.4.1 presents the key challenges related to the management of a cloud environment. Second, Section 2.4.3 presents the specific resource allocation challenges related to managing VNEs. Subsequently, Section 2.4.4 focuses on the challenges related to managing NSs in NFV-enabled cloud environments. Third, Section 2.4.5 proposes a taxonomy of the related work on resource allocation approaches in cloud environments. Finally, Section 2.4.7 concludes this chapter with emerging research directions in this field.

2.4.1 Cloud management

According to Jennings et al., the two main cloud management functions required to successfully operate a cloud infrastructure are *application placement* and *migration control* [16]. These functions are needed to optimize the operation of the cloud with respect to its management objectives or constraints. These goals and constraints can be related to the management of the cloud itself, or they can be related to the SRs. Important management concerns related to the cloud infrastructure are minimization of resource fragmentation, energy and resource consumption. These concerns are all in some way related to the scarcity of resources. The concerns related to the services/applications include both functional and non-functional aspects. The objectives related to the services are typically described in the agreed SLAs between the cloud provider and cloud user.

2.4.1.1 Application placement

Application placement entails deciding whether a request for resources is accepted or declined, referred to as *admission control*; and deciding how to realize this request using the cloud infrastructure, referred to as *placement control*. Placement control entails finding a good initial PC for an application, which requires solving the Application Placement Problem (APP). One can distinguish between the offline and online APP. The offline APP is the task of finding an initial PC for a set of requests concurrently. The online version considers each request separately. In practical cloud environments, the requests typically arrive one at a time. Competitive analysis is a method invented for analyzing online algorithms, in which the performance of an online algorithm is compared to the performance of an optimal offline algorithm that can view the sequence of requests in advance. An online algorithm is competitive if the ratio between its performance and the offline algorithm's performance, is bounded. Hence, competitive algorithms are used to overcome uncertainties about the future.

Initial approaches to the Cloud Application Placement Problem (CAPP) considered the placement of VMs, without considering the communication required between VMs. This approximation can be justified for deployments within a single datacenter, where the network is rather well-controlled and can be heavily over-provisioned. In this case, when the goal is to host as many requests as possible, the VM placement problem can be reduced to the Multidimensional Knapsack Problem (MKP) [38]. The MKP is known to be NP-hard. In geo-distributed clouds however, this approximation is no longer valid and the placement must be network-aware for the following reasons. First, the bandwidth requirements between VMs are often more limiting than the nodal requirements. Second, there often is a maximum acceptable delay for the communication between VMs. Network-aware approaches are closely related to the VNE problem, discussed in Section 2.4.3.

2.4.1.2 Migration control

Migration includes all adjustments made to PCs. These adjustments are made for one of the following reasons.

- *Fragmentation of SN's resources:* Since the requests for resources arrive one at a time and the resources in the SNe are scarce, the PC of one request can complicate the placement

of a later one. Further, when one request is terminated, the PC of the remaining request might be altered in order to reduce energy consumption. For instance, VMs could be collocated, allowing powering down a PM.

- *Changes in the SRs:* Over a service's lifetime, the demand for it can vary. The cloud user can deal with these variations, through a combination of horizontal and vertical scaling. In case of vertical scaling, when the capabilities of the SNo do not allow the dimensions of the VNo to increase any further, the management can decide to migrate the VNo to another SNo.

- *Changes in the SNe:* The characteristics of the SNe can vary over time. The cloud management must be able to react to this dynamicity. For instance, it can decide to migrate a VNo out of a failing SNo. In case the observed delay for VL becomes too high, e.g., because the loading of the PLs along its PP in the SNe increases, the manager can decide to reroute the VL.

2.4.2 Resilience in cloud computing

Reliability is an important non-functional requirement, as it outlines *how* the software systems realizes its functionality [39]. It refers to the system's capacity to recover from failure. Cloud computing enables highly available services, running on top of commodity hardware. To provide resilience in face of software and hardware failures, there are two types of schemes. First, there are schemes of *protection*. These schemes aim to prevent or limit service outage by introducing redundancy into the application's PC, prior to failure happening. When failure happens, the protection scheme should render the failure transparent to its user. One can distinguish between $active/active$ protection schemes, which have primary and backup resources active at the same time, and $active/passive$ protection schemes, that only activate the backup resources when a failure has happened. Compared to $active/active$ schemes, $active/passive$ schemes typically require fewer resources, since backup resources can be pooled together between multiple services. This overhead reduction does come at the cost of increased downtime and data loss. Second, there are schemes of *restoration*. Restoration schemes are reactive, in that they do not foresee any backup resources upfront. In response to failure events, service recovery is attempted by migrating the virtualized resources out of the failed substrate resources. Restoration schemes cause the least overhead but tend to result in the longest service downtime and the most data loss.

Jayasinghe et al. model cloud infrastructure as a tree structure with arbitrary depth [40]. Physical hosts on which VMs are hosted are the leaves of this tree, while the ancestors comprise regions and availability zones. The nodes at bottom level are physical hosts where VMs are hosted. Wang et al. were the first to provide a mathematical model to estimate the resulting availability from such a tree structure [41]. They calculate the availability of a single VM as the probability that neither the leaf itself, nor any of its ancestors fail. Their work focuses on handling workload variations by a combination of vertical and horizontal scaling of VMs. Horizontal scaling launches or suspends additional VMs, while vertical scaling alters VM dimensions. The total availability is then the probability that at least one of the VMs is

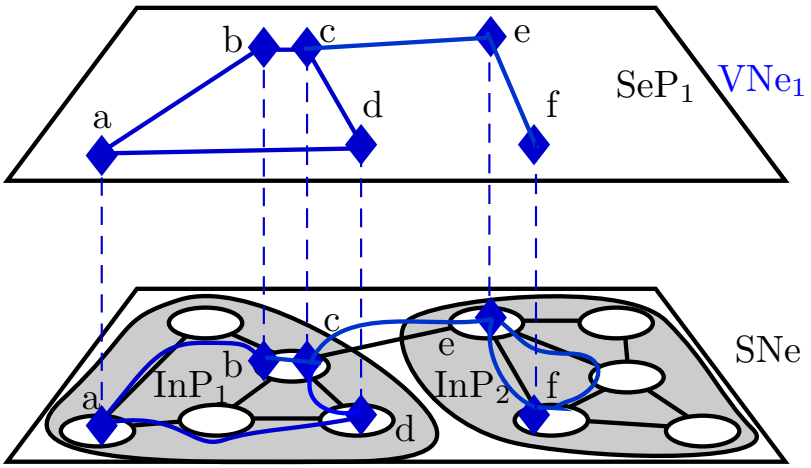


Figure 2.4: Illustration of VNE.

available. While their model suffices for traditional clouds, it is ill-suited for a heterogeneous cloud environment as link failure and bandwidth limitations are disregarded.

Further, there are works that model data availability in replicated file systems [42, 43, 44, 45, 46]. These models are dynamic, in that they consider the impact of the time between failures and the time to recovery. While these works can be used to estimate the impact of system parameters and replication schemes, they lack algorithms to synthesize a good replication scheme that is optimized for the required QoS-level and the cloud characteristics.

2.4.3 Virtual Network Embedding

The VNE problem is the task to embed a VNe onto a SNe, which is illustrated in Figure 2.4.

VNE entails both the task of node mapping, i.e. assigning each VNo to a SNo and link mapping, i.e. assigning each VL to one or more PPs in the SNe. These tasks depend on each other, as a VL instance from one VNo to another, must be routed from the SNo onto which the former VNo is mapped, to the SNo onto which the latter is mapped.

A PP is an ordered list of PLs. When the SNe supports path-splitting, the VL bandwidth can be distributed over multiple PPs, originating and terminating at the same source and target SNo, respectively. In Figure 2.4, the flow along the VL instance between VNos e and f is split over two disjoint PPs in the SNe. One of these PPs introduces one hop, while the other introduces two hops. Typically, there is an upper limit on the tolerable delay-difference between the PPs over which a single VL is split. For an overview of VNE, the reader is referred to [47]. The link mapping must respect the PL bandwidth capabilities of each PL. The total bandwidth consumption along a PL is the sum of the bandwidths of all VLs that are routed over this PL. The node mapping considers the resource requirements of the VNos and the capabilities of the PMs. Finding an optimal solution to the VNE problem is NP-hard [48]. Optimal solutions can be found using an ILP formulation, e.g. the one in [49, 50].

Fischer et al. distinguish between two types of heuristic strategies to the VNE [47]. First,

there are approaches that do not coordinate the node and link mapping. For instance, Razzaq et al. first try to map the request's VNos in order of decreasing processing requirements [51]. In the second phase, if all VNos are assigned, then the required VLs are allocated. To route a VL between two PMs, the k shortest PPs in the SNe are traversed in order of increasing path-length. The VL is assigned to the first PP with enough remaining bandwidth. Since the two stages are uncoordinated, a poorly chosen node mapping, will often complicate the link mapping. Especially, when well-connected VNos are assigned to poorly connected SNos.

Second, there are approaches that do coordinate these two subtasks. Coordinated approaches can either perform node and link mapping in 1 or in 2 separate stages. On the one hand, Cheng et al. propose a coordinated algorithm that first maps VNos to SNos in order of decreasing rank and then routes the VLs between the mapped nodes [52]. They consider two routing scenarios. In case of splittable flow, the Linear Program (LP) formulation of the Multi Commodity Flow (MCF) problem is solved. In case of unsplittable flow, for each VL, the Shortest Path (ShP) with enough remaining bandwidth is selected. A VNo's rank is based on both its resource and topological attributes. The ranking results in a node mapping, that tries to map well-connected VNos to well-connected PMs. On the other hand, in the same paper [52], Cheng et al. propose a 1-stage algorithm that maps the VNos to SNos in order of decreasing rank. The key difference is that this algorithm performs the node and link mapping at the same time. Each time a VNo is mapped, its required VLs are immediately routed using a Breadth First Search (BFS) algorithms. Lischka et al. propose another 1-stage coordinated algorithm, that is based on subgraph isomorphism detection [53]. Their proposed algorithm in each step maps a VNo to a PM and maps the VLs that are targeted at the current mapping simultaneously. Coordinating the node and link embedding can significantly reduce the embedding cost.

The SVNE problem is the task of embedding a VNe, so that it can survive one or more failures in the SNe [54]. For instance, Rahman et al. propose two strategies to deal with a failing PL [55]. First, they propose a restoration approach that calculates candidate detour paths for each PL. When after initial placement a given PL fails, then the VLs that use this PL are rerouted. Second, they propose a proactive approach, which prereserves both primary and backup bandwidth for each VL on link disjoint PPs. While the aforementioned approach considers only a single PL to fail, SiMPLE can protect against multiple concurrent PL failures by splitting the primary and backup flows over multiple disjoint PPs in the SNe [56]. For instance, when a flow requiring x bandwidth units is routed over 3 disjoint PPs, then provisioning $x/2$ and x bandwidth units along each PP results in a SVNE that can survive 1 and 2 PL failures, respectively. Finally, DRONE protects against a single PM or PL failure [57]. It does so by placing a primary and backup VNE on two disjoints subgraphs in the SNe.

2.4.4 NFV service orchestration

In order to implement the NFV orchestrator shown in Figure 2.3, service orchestration algorithms are needed. These orchestration algorithms realize two important functions [58, 59], which are illustrated in Figure 2.5.

First, the *chain composition* when the orchestrator composes the VNF-FG based on the SRs. This VNF-FG describes the logical structure of the application, i.e. it entails the VNF instances that realize the service and their required interconnections, i.e., VL instances. Second, the

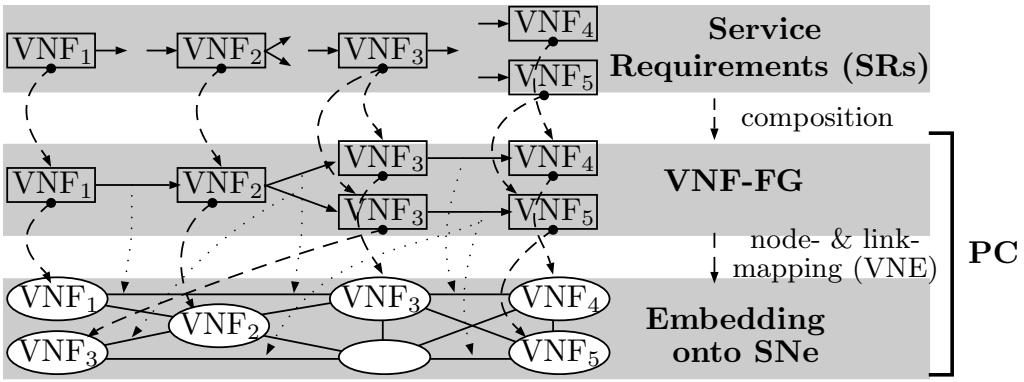


Figure 2.5: Illustration of the key service orchestration concepts in NFV environments.

service *embedding* where the orchestrator decides on how to embed the virtualized resources that constitute the VNF-FG onto the SNe. The embedding entails the task of node mapping, i.e. assigning the VNFs in the VNF-FG to SNos; and the task of link mapping, i.e. assigning the VL instances between VNF instances to PPs in the SNe. This process is closely related to the VNE problem, discussed in Section 2.4.3. The combination of a composed VNF-FG, together with an embedding of this VNF-FG onto a SNe is referred to as a PC. Since the VNE problem is NP-hard, the combined service orchestration problem is at least NP-hard.

2.4.5 A taxonomy of resource allocation approaches

This section provides an overview of the categories into which the related work is subdivided.

2.4.5.1 Static vs. dynamic

On the one hand, *static* (S) approaches do not adjust the placement of a request, once it is placed. On the other hand, *dynamic* (D) approaches do consider adjustments to the PC of deployed requests.

Since migration is an operation requiring a lot of overhead, dynamic approaches typically consider a migration cost that incorporates the required migration bandwidth, processing resources or service downtime.

2.4.5.2 Strategy

- *Exact (E):* when these algorithms return a solution, it is guaranteed to be optimal in the objective function. These solutions are typically based on an ILP formulation.
- *Approximation (A):* these algorithms can find solutions with guarantees on the optimality of the solution.
- *Heuristic (H):* these solutions can find a solution faster but provide no guarantees on the optimality of the solution.

- *Model (M):* these works do not propose resource allocation algorithms but model the behavior of the system.

2.4.5.3 Application topology

The existing orchestration approaches consider one of the following application topologies.

- *Set of tasks:* the workload comprises a set of tasks that must be scheduled on a set of PMs, often considering a maximum execution time for the entire application. For instance, Lee et al. consider the initial distribution of tasks with a fixed processing time over a set of PMs and subsequent redistribution to reduce the total energy requirements through task consolidation [60]. Since the PC is modified after initial deployment, the approach is labeled 'Dynamic'. In contrast, Zhong et al. consider the problem of finding an initial scheduling for a set of tasks, while minimizing the total execution time and maximizing the load balancing [61]. Since they only consider the initial task assignment, their approach is labeled 'Static'. In their model, the processing time for a task depends on the instruction count of the task and the processing capabilities of the PM on which it is executed, expressed in instructions per second. The major limitation of this work is that the communication between tasks is not considered.

- *Set of VMs:* the application comprises a set of VMs; the communication between these VMs is not considered. An important difference compared to approaches that consider a set of tasks, is that while the tasks of an application can be executed sequentially, all VMs required for an application are typically considered active over the application's lifetime. An example of such an approach is the work by Camati et al. that approaches the VM placement problem as an MKP [62].

- *Set of replicas:* Silberstein et al. propose a 'lazy' replication scheme for EC replicated filesystems that postpones recovery operations. By grouping these recovery operations, recovery bandwidth is saved, at the cost of an increased data loss probability [43].

- *Graph:* the embedding of a generic graph onto a SNe is the VNE problem. For instance, Jarray et al. consider the problem of embedding VNe graphs [63]. The authors propose a batch approach based on column generation, combined with an auction technique to price resources. Each column represents a possible PC for a VNe. Their proposed heuristic can significantly improve the acceptance ratio, by using the SNe resources more efficiently.

- *Replicated graph:* the VNF-FG required for the NS is allocated more than once. For instance, DRONE protects against single PM and PL failures by placing a primary and backup embedding on two disjoint subgraphs in the SNe [57].

- *SFC:* the VNF-FG topology entails a set of NFs that form a directed acyclic path. For instance, Bari et al. propose an exact ILP and heuristic based on DP to embed such an SFC [64].

- *DAG:* the topology is restricted to be a connected graph that does not contain any cycles. For instance, the virtualized CDN topology considered by Bouten et al. is a hierarchical

caching network comprising a single inner core Points of Presence (PoP). Its content is delivered to multiple access PoPs through a sequence of outer core, and aggregation PoPs [65].

- *Tree:* a tree is an undirected graph in which any two vertices are connected by exactly one path. Typically, the VNF-FG is a directed rooted tree. Ocampo et al. propose a Mixed Integer Linear Program (MILP) [66] that can compose the minimal bandwidth VNF-FG tree. Each SFC from the single root VNF towards any of the terminal VNF-instances satisfies the chaining requirements of the composing VLs, which are part of the SRs. The service model considers that the VNFs bandwidth requirements depend on the order in which the VNFs are chained. Further, since the algorithm can decide on the best order of VNFs in this tree, the number of instances of each VNF depends on the composition. The major limitation of this work is that the authors do not consider the embedding of this VNF-FG.
- *Cactus graph:* it is a connected graph in which every edge belongs to at most one simple cycle. For instance, Rost et al. propose a decomposition scheme for cactus graphs. Their algorithm is based on this reformulation and a LP relaxation of the problem [67]. It is the first polynomial time algorithm with guarantees on the optimality of the resulting solutions. Since the algorithms' probabilistic guarantees on profitability and capacity violations are rather relaxed, its contributions have little practical use.

2.4.5.4 Composition

Some works assume the VNF-FG to be precomposed, hence the composition task is not considered. Others do consider the VNF-FG composition. These approaches can be classified using the following criteria.

- *Flexibility in the sequence of the NFs (seq):* the order of the VNF instances in the VNF-FG can be varied. For instance, Ocampo et al. consider the optimal chaining of VNFs in a directed tree topology [66]. The flow on each egress VL, must pass through a set of VNFs, prior to termination in a terminal VNF. For a VNF instance, the bandwidth for each of its egress VL instances is proportional to its total ingress bandwidth.
- *Dependence of resource requirement on PC (res):* the resource requirements of a VNF instance depend on the composition of the VNF-FG and on the applied workload. For instance, Ma et al. consider the problem of routing SFCs through middle-boxes [68]. The authors consider flexibility in the order in which the SFCs flow through the middle-boxes. The egress bandwidth of a VNF is assumed proportional to its ingress bandwidth, by the associated traffic changing factor. Therefore, the bandwidth requirement of a VNF instance depends on its order in the SFC. The authors propose a DP that can decide on the order of the middle-boxes in the SFC. The major drawback is that their model assumes the processing requirements independent of the ordering of the SFCs.
- *Optional NFs (opt):* not all VNFs are strictly required to be part of the VNF-FG. Bouten et al. consider the problem of installing a caching hierarchy [65]. In their model, the ingress bandwidth to a cache depends on the cache's hit rate, determined by its size, and

its egress bandwidth. Both the number of instances in each caching layer and the size of each cache is flexible.

- *Flexibility in the number of instances (inst):* the number of instances of a VNF in the VNF-FG depends on the composition and/or the workload. For instance, Houdi et al. consider the problem of adapting the PC of an existing request, to deal with a changing workload [69]. They propose an exact algorithm based on an ILP formulation of the problem and a greedy heuristic that can adjust the PC through a combination of horizontal and vertical scaling and migration.

2.4.5.5 Embedding

Some approaches only consider the placement of VNos onto PMs and disregard the traffic between the VNos. For instance, Tordsson et al. propose an ILP for the VM placement problem [70]. They try to maximize the resource utilization, subject to a maximum total deployment cost. Contrastingly, authors in [71] consider the problem of assigning n VMs to n different PMs, while minimizing the routed bandwidth. They consider a routing cost based on predefined SNe routes. The proposed heuristic tries to assign VM-pairs with heavy mutual traffic to PM-pairs with low-cost connections. First, it clusters the PMs, using the hop-distance between them as the partition criterion. Then, the VMs are partitioned into k VM-clusters with minimum inter-cluster traffic. The major limitation of this work is that the routing is assumed predefined and PL bandwidth capabilities are not considered.

Other works do consider the VL routing. For instance, Calheiros et al. propose a 2-stage VNE heuristic [72]. In the first stage, the node mapping is performed in order of decreasing processing requirements. After initial node assignment, a local search is performed on the node mapping to improve the load balancing. In the second stage, the link mapping is performed in order of decreasing bandwidth requirements. The routing is performed using the A*-ShP heuristic.

2.4.5.6 Objective function/constraints

An aspect related to the quality of a PC can be considered in the objective function, and/or as a constraint. In case an aspect is considered only in the objective function, it is indicated with 'O' between parentheses. When this aspect is considered as a constraint, then it is indicated with 'C' between parentheses. If an aspect is modeled and not used in a management algorithm, then it is indicated with 'M' between parentheses. Following types of aspects, related to the PC are considered.

- *Nodal (nod):* these are requirements related to the VNos. For instance, Virtual Machine Usage Based Placement in IaaS Cloud (VUPIC) rearranges VMs according to their Resource Usage Vector (RUV) [73]. This 3-dimensional RUV considers CPU usage (percentage), network utilization (bytes per second and disk I/O (bytes per second). The main objective of VUPIC is to club together those VMs which consume different resources, to minimize resource contention.
- *Bandwidth (bw):* these are related to the bandwidth requirement between VNos, or the routed bandwidth in the SNe. For instance, the recursive VNE heuristic proposed by

Lischka et al. considers the residual bandwidth capabilities on each PL in the SNe when it generates the possible candidates to add to the current mapping [53]. It uses a BFS procedure to construct a PP with enough remaining bandwidth for each VL between the candidate VNo and the already mapped VNos.

- *Availability (ava):* these are related to the availability of the service. For instance, Jiang et al. consider the SVNE problem with availability constraints on individual VLs and VNos [74]. They propose an ILP that can replicate VLs and VNos. The major limitation of their availability model is that it does not support overall service availability constraints and failure correlation. Further, each PL and PM can be used only once for each request. Other approaches consider the impact of workload variations on service availability. For instance, Wang et al. [41] consider both workload and PM availability in their availability model. The authors minimize bandwidth consumption by a combination of horizontal and vertical VM scaling.

- *Delay (del):* these are related to the maximum tolerable delay for a VL or the maximum end-to-end delay between VNos. For instance, Inoue et al. consider a maximum acceptable latency for VL instances [75]. The delays in the SNe vary according to the loading of PMs and PLs. They propose a distributed approach that minimizes the migrations caused by uncertainty in the delays.

- *Energy (ene):* these are related to the energy required to place the request. These placement approaches build on an underlying energy consumption model. For instance, Marotta et al. consider the problem of embedding a set of SFCs with minimal total power consumption, while considering uncertainty in the CPU consumption of VNos [76]. They formulate the problem as a robust optimization problem that considers the power profile of the servers and switches. When a switch or server is powered down, it does not consume any power. When it is powered on, then its power consumption is the sum of the idle power consumption and a term proportional to the loading of the component.

- *Cost (cos):* these are related to the costs incurred by the provider to deploy the NS. For instance, Ghribi et al. consider the problem of embedding an SFC with minimal cost [77]. The authors consider heterogeneous hosting and routing costs throughout the SNe. They propose a heuristic algorithm based on DP.

2.4.6 State of the art

An overview of the related work on resource allocation in cloud environments is shown in Table 2.2.

Table 2.2: Overview of the related work on resource allocation.

Ref.	S/D	Strat.	Contribution	Shortcoming	Top.	Comp.	Emb.	O/C/M
[65]	S	E, H	ILP formulation and a GA. Consider the effect of cache size and hit rate on the required bandwidth and expected latency.	The embedding of nodes and links is fixed. Only an average end-to-end delay is considered.	DAG	yes: opt, res	VL, VN	nod(C), del(C), bw(O), ene(O), cos(O)

Table 2.2: Overview of the related work on resource allocation.

Ref.	S/D	Strat.	Contribution	Shortcoming	Top.	Comp.	Emb.	O/C/M
[69]	D	E, H	An exact ILP and a greedy heuristic that can adapt the PC of a service to changing workload requirements through a combination of horizontal and vertical scaling and migration.	The approach does not consider any flexibility in the order of the VNFs.	DAG	yes: res, inst	VL, VN	nod(C), bw(C), cos(O)
[53]	S	H	Recursive algorithm based on subgraph isomorphism detection: it maps nodes and links during the same stage. It can find a feasible solution fast.	The algorithm stops as soon as a feasible solution is found, without any guarantees on optimality.	Graph	no	VL, VN	nod(C), bw(C)
[49]	S	E, H	An exact MILP and heuristics based on reformulation of the problem on an augmented graph and randomized and deterministic rounding.	Collocation within a service is not allowed.	Graph	no	VL, VN	nod(C), bw(C), cos(O)
[78]	S	E, H	Reduction of LC-VNE to the minimum-cost maximum clique problem and proposal of two heuristics.	Each PL and PN can only be used once for each service.	Graph	no	VL, VN	nod(C), bw(C), cos(O)
[75]	D	H	Application of a Yuragi-based method that reduces the number of migrations needed to satisfy the maximum end-to-end latency while considering the effect of requirement fluctuation and topological SNe change.	The VL delay calculation only considers the delay contribution by the highest-loaded PL on the PP.	Graph	no	VL, VN	nod(C), del(C), bw(C)
[79]	S	H	An MCTS algorithm that can find a better solution when given more time.	Absent a good roll-out strategy, the algorithm performs a random search.	Graph	no	VL, VN	nod(C), bw(C), cos(O)
[56]	S	E, H	Two methods to protect against single link failures: a PL and path based protection method. The approach reserves backup bandwidth.	Node failures are not considered.	Graph	no	VL, VN	nod(C), del(C), bw(C), ava(C)
[63]	S	H	A near-optimal offline VNE heuristic based on column generation that can improve the resource utilization by pricing the SNe resources using an auction technique.	The pricing of the SNe resources does not consider the lifetime of each VNe request, reducing the resource utilization.	Graph	no	VL, VN	nod(C), del(C), bw(C), cos(O)
[74]	S	E, H	An exact ILP and two heuristics that can solve the SVNE problem with availability requirements on each VNo and VL instance.	The availability requirements are not end-to-end. Failures are considered uncorrelated.	Graph	no	VL	nod(C), bw(C), ava(C), cos(O)

Table 2.2: Overview of the related work on resource allocation.

Ref.	S/D	Strat.	Contribution	Shortcoming	Top.	Comp.	Emb.	O/C/M
[72]	S	H	A two-stage VNE heuristic that performs a local search during the node-mapping phase in order to improve load balancing.	The node- and link-mapping phases are not well-coordinated.	Graph	no	VL	nod(C), del(C), bw(C)
[71]	S	E, H	Reduction of the traffic-aware VM placement problem that minimizes the SNe bandwidth to the QAP.	Collocation of VMs and bandwidth limitations are not considered.	Graph	no		nod(C), bw(O)
[57]	S	E, H	An exact and heuristic algorithm that protect against single PM and PL failures by placing two copies on disjunct subgraphs of the SNe.	Simultaneous failure and heterogeneous failure behavior are not considered.	Replicate graph	no	VL	nod(C), bw(C), ava(C), cos(O)
[45]	D	M	Markov chain model of replica loss and replica repair that considers storage and bandwidth limits.	Lacks a replica management algorithm.	Set of repli-cas	no		nod(M), bw(M), ava(M), dur(M)
[43]	D	M	A replication scheme that postpones recovery of failed data chunks to reduce recovery bandwidth.	Lacks a replica management algorithm.	Set of repli-cas	yes: inst		nod(M), bw(M), ava(M)
[80]	S	E	A DP that maximizes a file's availability, while guaranteeing a maximum replication cost and nodes with heterogeneous availability and cost.	PL failure and nodal resource limitations are not considered.	Set of repli-cas	yes: inst		nod(C), ava(O)
[81]	S	H	A replication strategy that distributes replicas over a cluster based on a non-similarity metric and minimizes recovery time and bandwidth, while guaranteeing a minimum availability and considering heterogeneous failure probabilities.	Correlation between failures and bandwidth limitations are not considered.	Set of repli-cas	no		nod(C), del(O), bw(O), ava(C)
[60]	D	H	Two energy-conscious task consolidation heuristics, which aim to maximize resource utilization and explicitly take into account both active and idle energy consumption.	The algorithms do not consider migration costs and communication between tasks.	Set of tasks	no		nod(C), ene(O)
[61]	S	H	A greedy Partical Swarm Optimization (PSO) algorithm that reduces the total execution time and improves the load balancing.	The communication requirements between tasks are not considered.	Set of tasks	no		nod(yes), del(O)
[62]	S	E, H	Approach the VMP problem as an MKP.	Communication between VMs is not considered.	Set of VMs	no		nod(C), cos(O)

Table 2.2: Overview of the related work on resource allocation.

Ref.	S/D	Strat.	Contribution	Shortcoming	Top.	Comp.	Emb.	O/C/M
[70]	S	E	An ILP for VM placement that maximizes the resource utilization, subject to a maximum prize for the total deployment.	The bandwidth requirements between VMs are not considered.	Set of VMs	no		nod(C), cos(C)
[73]	D	H	Improvement in VM performance isolation in multiple dimensions and overall resource utilization through usage-based migration.	The PL bandwidth limitations are not considered.	Set of VMs	no		nod(C), bw(C)
[41]	S	H	Consider both workload and PM reliability in their availability model. Minimize bandwidth consumption by a combination of horizontal and vertical VM resizing.	The algorithm does not consider PL failure.	Set of VMs	yes: res, inst		bw(O), ava(C)
[82]	S	E, H	An exact MILP and a heuristic that employs a binary search to maximize the number of SFCs that can be accepted at the same time.	The heuristic does not scale very well as it involves solving an MILP several times	SFC	yes	VL	nod(C), del(C), bw(C), cos(O)
[64]	S	E, H	ILP formulation and a heuristic based on DP.	Multiple VNFs of the same service cannot be collocated	SFC	no	VL	nod(C), del(O), bw(C), ene(O)
[77]	S	E	DP that supports collocation of VNFs of the same service.	The order of the VNFs in the SFC is fixed.	SFC	no	VL	nod(C), bw(C), ene(O), cos(O)
[83]	S	E, H	A competitive online algorithm.	PL bandwidth is not considered.	SFC	no	VL	nod(C), del(C), cos(O)
[76]	S	E	Application of the theory of robust optimization to deal with uncertainty in the input parameters.	The optimal algorithm does not scale well for larger problem instances.	SFC	no	VL	bw(C), ene(O)
[68]	S	H	A DP that optimizes the order in which an SFC passes through a series of middle-boxes.	The NF placement is not considered. Processing requirements are not considered.	SFC	yes: seq, res	VL	nod(O), bw(C), cos(O)
[84]	S	E, H	An exact ILP and a heuristic based on LP relaxation that can chain and embed SFCs that have a partially ordered VNFs.	The bandwidth requirements are considered constant through the SFC.	SFC	yes: seq	VL	nod(C), del(C), bw(C), cos(O)
[67]	S	E, A	The first polynomial time service chain approximation algorithms both for the case with admission and without admission control.	While the algorithm provides guarantees on the quality of the embedding, other heuristics perform better.	SFC/ Cacti	no	VL	nod(C), cos(O)

Table 2.2: Overview of the related work on resource allocation.

Ref.	S/D	Strat.	Contribution	Shortcoming	Top.	Comp.	Emb.	O/C/M
[85]	S	H	A recursive heuristic that coordinates composition and embedding.	The algorithm stops as soon as a feasible solution is found.	Tree	yes: seq, res, inst	VL	nod(C), bw(C)
[86]	S	H	One of the first papers that considers both the composition and the embedding problem. A heuristic that greedily minimizes the VNF-FG bandwidth and embeds it using an MIQCP	The composition and embedding stages are not coordinated.	Tree	yes: seq, res, inst	VL	nod(C), del(O), bw(C), cos(O)
[66]	S	E	A first MILP that can compose the exact minimal bandwidth VNF-FG.	The algorithm does not consider the embedding stage.	Tree	yes: seq, res, inst		bw(O)

2.4.7 Emerging research directions

This section highlights the future research directions of resource allocation in cloud environments. Three main directions that may propel the research in this field in the near future are identified. Replica management in very dynamic cloud environments; availability-aware SVNE; and network-aware NFV orchestration.

2.4.7.1 Dynamic replica management

There is a need for outcome-oriented service management, since in a public cloud the provider is remunerated based on the service outcome, according to the agreed SLA. To maximize his revenue, the provider must consider the impact of his management decisions on the service outcome and the costs that he incurs to provision the service. When the service outcome is acceptable, then the provider receives a reward. In case of an unsatisfactory QoS the provider typically pays a penalty.

The trend towards geographically decentralized computing environments, means that unreliable and resource-constrained hardware is increasingly being included into the cloud infrastructure and that both hardware and services are increasingly becoming heterogeneous. Therefore, provisioning fault-tolerant services in a cost-effective way is becoming increasingly challenging. The cloud provider must continuously monitor the service and the SNe and decide on the appropriate replication actions to take. There have been various studies that model the impact of the time between failures and the distribution of recovery times on the availability of data in replicated (file) system [44, 45, 42]. These studies were typically based on overly simplistic failure models or focused on analytic calculation of a metric with very limited practical meaning, namely the Mean Time to Dataloss (MTDL), which can be in the order of a 1000 of years. The provider however, is not directly interested in these metrics, but rather in assessing the impact that replication decisions have on the service outcome. Hence, we conclude that further research is needed into resource allocation algorithms that can provide intelligent replication decisions over a service's lifetime.

2.4.7.2 Availability-aware SVNE

Current SVNE approaches lack an availability model. They typically focused on finding an embedding that can survive 1 or more SNo or SLs failures [56, 57]. When the failure behavior in the SNe is homogeneous, this can be a good first approach. However, they do not provide guarantees on the resulting availability of the SVNE. In geo-distributed cloud environments, the failure behavior can be very heterogeneous. This means that some parts of the infrastructure can be fairly reliable, for instance the part in a datacenter, while the PMs and PLs at the network edge are very unreliable. The existing approaches use the same level of protection anywhere in the SNe. They cannot intelligently adjust the protection level to the resilience requirements of the service. Additionally, current SVNE approaches typically require that virtualized resources of the same application cannot be collocated on the same PM. However, for a given protection level, it is well-known that the total failure probability increases as the number of independently failing devices in the SNe increases. Therefore, given the recent trend towards geo-distribution, intelligent SVNE algorithms are needed that allow for more flexibility in the PC and that provide availability guarantees.

2.4.7.3 Network-aware NFV orchestration

On-demand service orchestration is an important requirement for the management of NFV environments. On the one-hand, on-demand service-orchestration requires the composition of the VNF-FG, based on the SRs. On the other hand, it requires the embedding of this VNF-FG. Most of the current orchestration approaches typically assume the VNF-FG precomposed [64, 77, 67, 83]. When the composition task is considered, the service model is too restrictive. Typically, the VNF-FG is restricted to be an SFC [68], or a tree [66, 86]. Hence, each VNF instance can have at most 1 ingress VL instance. Given the rise of novel broadband services that interconnect multiple sources and targets on-demand, e.g., Twitch [11], there is need for a service model that can produce more general VNF-FGs. Further, current orchestration approaches that do consider both composition and embedding tasks, perform both tasks in two separate stages [86]. In other words, the VNF-FG is composed without considering the availability of resources in the SNe. In IaaS clouds, the infrastructure provider has control over both the cloud infrastructure and the NS. Hence, the cloud provider should optimize the composition and the embedding at the same time. Clearly, there is need for NFV orchestration algorithms that perform both tasks in a coordinated way.

2.5 Conclusions

The demand for real-time broadband services continues to surge. Today's services are increasingly demanding in terms of throughput and response times. Therefore, while cloud computing initially centralized computing resources in remote datacenters, now there is a need for storage, compute and networking resources close to the end user. Today, VoD service providers, such as YouTube and Netflix partner up with ISPs to install private CDNs at the network edge. These caches are typically provisioned based on historical demand. In the near future, applications will continue to generate larger amounts of data and demand lower response times. In order

to enable mission-critical IoT applications, such as cloud robotics, Industry 4.0, self-driving cars and Cloud-Radio Access Network (C-RAN), a wide variety of tasks must be executed near the edge of the network. Incorporation of these devices into the cloud formalism will enable service providers to share these resources in a flexible, cost-effective way. At the same time, these devices severely complicate the management of the cloud environment. The capabilities of the infrastructure at the edge are very limited and the infrastructure is much less reliable than the hardware in datacenters. The cloud management must therefore carefully balance the desired service outcome and the resource consumption. Current cloud management algorithms cannot deal with this highly heterogeneous infrastructure. The orchestration of realtime NSs that can hardly tolerate any downtime at all is particularly challenging on best-effort hardware. Therefore, SVNEs are needed that can protect the service against a combination of SNo and SL failures. For these realtime services, the desired outcome is the aggregate service availability. Current SVNE algorithms are not availability-aware and therefore further research is needed. Further, data persistence is very important for many IoT and big data applications. Given the scarcity of resources at the network edge, storage at the edge will be temporary. In order, to balance storage cost and data loss, SLA-aware resource allocation algorithms are needed that can synthesize good replication strategies.

Tomorrow's IoT applications have diverse, often conflicting QoS requirements. To optimize wireless communications on a per-service basis, for objectives such as power consumption, throughput and ultra-low latency, the RAN will be virtualized in 5G. In order to deploy these services, on-demand service orchestration algorithms are needed that can compose services with complex topologies in NFV environments. Since the sources of these services are no longer provided by the service provider itself and traffic aggregation will become more important, the orchestration algorithms must be able to compose VNF-FGs that aggregate traffic. Further, these compositions must be made on-demand and consider the SNe resources, while keeping the execution time of the orchestration algorithms under a few seconds. Current orchestration algorithms do not support traffic aggregation and have poor or no coordination between the composition and embedding subtasks.

Chapter 3

Cost-effective replica management

This work was supported by the University of Antioquia and by the FUSE project.
The underlying ideas have been published in [C6].

This chapter focuses on the challenges related to the orchestration of storage resources in heterogeneous cloud environments. More specifically, it focuses on Question I. As introduced in Chapter 1 and 2, compared to centralized clouds, the storage capabilities at the edge of the network are much more limited. Hence, while centralized storage will be employed for long-term storage, storage at the network edge will be mainly temporary. Moreover, storage capabilities and connectivity vary strongly from one edge location to another. This chapter investigates how to replicate data effectively across storage nodes in heterogeneous cloud environments.

Cloud providers rely on fault-tolerance mechanisms to realize high-availability services on best-effort infrastructure. Service replication limits the data loss caused by failure, at the expense of additional operational costs. Recently, with the advent of MEC, cloud environments are becoming increasingly heterogeneous and dynamic, by the incorporation of (very) unreliable and resource-constrained devices. In this chapter, we investigate how to devise an economically viable replication strategy, for a given service on a particular cloud environment. Previous work either focused on finding replication strategies for stateless services, ignoring recovery processes and correlated failures, or considered system dynamics, while lacking SLA-awareness. We approach the replica management problem as a runtime revenue maximization problem. Our proposed DP algorithm can generate the optimal replication strategy over the application lifetime. Through extensive simulations, we show that our algorithm significantly improves provider revenue over a wide range of cloud- and SLA-conditions and adapt its strategy to evolving operating conditions. The results show that coupling dynamic failure models with SLA-awareness can lead to profitable replication strategies, even in cases where providers currently turn a loss.

3.1 Introduction

In this work, we are the first to introduce and analyze the GRMP, which is the problem of finding a replication strategy that maximizes expected revenue over the service lifetime. The GRMP approaches the problem of data loss prevention from the economical perspective of a service provider. The goal of this chapter is to find an optimal placement strategy over the envisioned service lifetime. Based on the current replication state and the remaining required lifetime, the replication algorithm decides whether scaling in, scaling out, or doing nothing is most cost-effective.

We identify following major contributions. First, we introduce the GRMP. Second, we propose an exact algorithm, which considers both a dynamic replication model and SLAs to maximize provider revenue. This algorithm can be used to generate optimal policies over a wide range of operating conditions. Third, we demonstrate that our approach can significantly improve provider revenue in both time-invariant and time-variant cloud environments, as our algorithm can adapt to varying system parameters.

The remainder of this chapter is structured as follows. Section 3.2 provides an overview of related work. In Section 3.3, the GRMP is introduced. We propose our solution in Section 3.4. Subsequently, we compare the performance of our algorithm to related works in Section 3.5. Finally, we conclude the chapter in Section 3.6.

3.2 Related work

SLAs describe the QoS-level that its users can expect. Nowadays, services can be anything ranging from VMs, containers and databases to disk images.

SLA violations can lead to hefty fines for service providers. To avoid these penalties, while relying on best-effort infrastructure (e.g., a datacenter), suffering from failures caused by faulty disks, software errors, power outage and network problems, providers employ fault-tolerance. In literature, there are two main protection schemes to make applications survive failures.

First, there are schemes of protection, which before any failures have happened, proactively instantiate additional resources. When failures happen, there are still sufficient resources available to make sure that the service remains available. Second, reactive schemes do not instantiate any additional resources before failure occurs but try to bring failed services back online by hosting them on different PMs. Reactive methods do not use any additional resources in the absence of failure. However, as bringing a new replica online always requires some minimal setup time, service outage cannot be avoided. Additionally, reactive methods cannot avoid that at least some data is lost. Therefore, reactive methods are typically only used to protect stateless services. In protection schemes a trade-off must be made between resource consumption and availability.

Considerable research effort has been devoted to the failure behavior in replicated systems. In a first approach, researchers have considered the failure behavior of services to be static. For instance, Jayasinghe et al. model cloud infrastructure as a tree structure with arbitrary depth [40]. Physical hosts on which VMs are hosted constitute the leaves of this tree, while the ancestors comprise regions and availability zones. The PMs at bottom level are physical hosts where VMs are hosted. Wang et al. estimate the availability of a single VM as the probability that

neither the leaf itself, nor any of its ancestors fail [41]. While static approaches can serve as a first approximation to assess the availability and hosting costs, they are not accurate as in reality failures exhibit both temporal and spatial correlation [87]. Static approaches rely on three very limiting assumptions. First, they assume a constant availability value for each machine over time. In reality, at some moment all machines seize permanently to function, consequently their instantaneous availability declines over time. Additionally, when a transient failure occurs it will take some minimum time before the machine is back online. Hence, temporal correlation in the availability of a single machine cannot be ignored. Second, these approaches typically assume the same set of machines to be used during the entire deployment. In contrast, replicated systems often migrate copies that are hosted on unrecoverable machines. These migrations will always take some time and induce migration costs, which are disregarded in static models. Third, in reality a stateful service becomes unrecoverable as soon as insufficient copies are online at a given moment. Hence, the availability of a single service is time-dependent as the probability that such a catastrophic event has occurred inevitably increases over time.

Therefore, more dynamic failure models are needed to estimate the total operation cost and service availability. Google researchers propose a time-homogeneous Markov model for replicated systems [42]. They define a cell as a pool of devices, together with their higher-level coordination processes. They show that replication across multiple racks, or even across geo-distributed clouds can be accurately modeled as multiple linked cells. Their model considers failure correlation, recovery and migration processes.

The most important limitation of the work is that they do not consider how to generate a "good" replication strategy, given a certain system model. Their availability model forms the basis for our proposed approach.

In distributed file systems two redundancy schemes are commonly used (1) replication, which creates identical replicas for each data block; and (2) EC, which transforms original data blocks into an expanded set of encoded blocks, such that any subset with enough encoded blocks can be used to reconstruct the original data blocks [88].

For system designers, it is important to have a replication strategy that addresses their specific challenges. The main types of resources that comprise the subjects of a cloud resource management system are compute, networking, storage and power [16]. In general, the provider's objectives related to resource consumption and the objectives of the cloud user conflict. For instance, Silberstein et al. focus on reducing the bandwidth required for the EC recovery process [43]. The authors propose a lazy recovery scheme: recoveries are queued and bundled, hereby reducing bandwidth consumption, while also increasing the failure probability. While the authors evaluate their proposed recovery schemes for a combination of disk, machine and rack failures, they do not provide a practical algorithm which can be used to select the best strategy. In another approach, Wang et al. try to maximize reliability and at the same time keep the hosting expenses under a maximum level [80]. However, the applicability of their approach is severely limited by the assumption of a static failure model.

The objectives of the providers center around efficient and effective resource use within the constraints of SLAs with the Cloud users [16]. Therefore, the providers need replication management algorithms that consider the impact of management decisions on operational expenses and the expected pay-off through SLA conformity. As these cloud environments are

increasingly changing over time, the replication management algorithms must become more dynamic.

Our approach exceeds the state of the art in that it focuses on the problem of revenue maximization during a certain time window. Other approaches are generally limited to finding analytic expressions for the MTDL, while lacking any notion of resource consumption costs. Moreover, our approach is not limited to an analysis of replicated systems but yields a practically usable replication management algorithm. Finally, our proposed algorithm can generate strategies that vary over the service lifetime.

In this chapter, we will focus on SLA violation as a direct result of failure. The impact of workload variations on response time requirements is considered out-of-scope. However, we will consider the indirect consequences of workload variations on data unavailability through their effect on system costs.

3.3 Generalized Replica Management Problem

In this chapter, we research how one should manage the Replication Level (RL) of a certain service over time, in order to maximize the expected revenue during its relevant lifetime. The remainder of this section is structured as follows. First, we discuss the replication model, its parameters, and the validity of the model. Finally, we provide a description of the GRMP.

3.3.1 Replication model

Google researchers showed that large-scale replicated systems can be modeled as time-homogeneous Markov processes [42]. In their model, the current behavior of the replicated system is uniquely determined by the current number of copies available. Hence, the process is memoryless, i.e. as time passes the process loses the memory of the past. They represent both failure and recovery processes by constant transition rates between states. Consequently, their approach can only be used to evaluate a fixed strategy. In our model, we use a time-inhomogeneous Markov chain. We observe the process periodically and assume that the behavior can only change at the start of a new period. An overview of the replication model parameters is given in Table 3.1. In the following we illustrate how these parameters can be determined in a Hadoop Distributed File System (HDFS) context.

L_{min} depends on the chosen replication scheme. When the number of active copies drops below this value, then the data is irrevocably lost. For naive replication the minimum RL is always 1, while for (k, n)-EC, this value is equal to k. L_{max} is the maximum number of copies that can be active at the same time. For naive replication, this value is only limited by the number of PMs available in a cell. However, to limit computation time its value can be set to a reasonable upper-limit and can then be further increased should a strategy with maximum RL be chosen by our algorithm. For EC, L_{max} equals n. Figure 3.1a shows an example for a single-cell replicated system with $L_{min} = 1$ and $L_{max} = 2$. In the model, the states, i.e. **I**, comprise two types. First, there are regular states (uncolored) for which the number of reserved copies and the number of active copies is the same. There is a state corresponding to RL 0, 1, ..., L_{max}. These states are also present in [42]. Second, we introduce action states (colored), which represent a decision to change the RL. In HDFS, the current RL can be queried

Parameter	Description
L_{min}	Minimum RL for the data to be recoverable.
L_{max}	Maximum RL to be considered.
$\mathbf{I_{ON}}$	States for which the data is recoverable.
$\mathbf{I_{OFF}}$	States for which the data is unrecoverable.
A_i	Set of possible actions (go-to states) from state i.
$C_{i,j}(n)$	Cost of action j in i, at the start of MI n.
$P_{i,j}(n)$	Transition probability from state i to j during MI n.
h	Management period, i.e. the time between MIs.

Table 3.1: Replication model parameters.

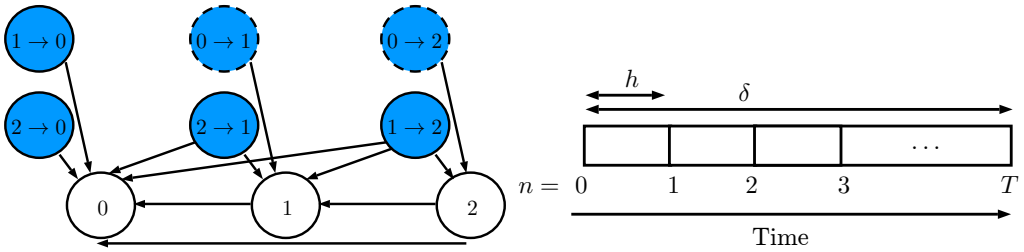
via the Application Programming Interface (API). The desired RL can be set via the same interface. The action costs can be estimated using advanced energy profiling methods, such as the one proposed in [89]. For practical purposes, it typically suffices to consider the hosting costs proportional to the RL. The migration cost can be approximated as proportional to the amount of data transferred through the network. In the following, we assume that scaling out takes a certain exponentially distributed time, while scaling in happens instantaneously. For a multi-cell configuration, the states and actions can be derived in a similar way.

The action states with an intermittent border represent actions, i.e., '$0 \rightarrow 1$' and '$0 \rightarrow 2$', which can only be taken upon the initial deployment of the service. These actions are assumed to take effect immediately, which means that the number of replicas at startup equals the desired number of replicas. After initial deployment, the data is either available or unrecoverable, depending on whether the state is in $\mathbf{I_{ON}}$ or $\mathbf{I_{OFF}}$, respectively. $\mathbf{I_{ON}} = \{1, 1 \rightarrow 2, 2\}$ and $\mathbf{I_{OFF}} = \{0\}$. $A_0 = \{0 \rightarrow 1, 0 \rightarrow 2\}$, $A_1 = \{1 \rightarrow 0, 1, 1 \rightarrow 2, 1 \rightarrow 3\}$, $A_2 = \{2 \rightarrow 0, 2 \rightarrow 1, 2\}$. When the service is in an action state at the beginning of an MI, then the only possible action is to stay in this same state. For instance, when there is one active replica and a second one is being generated, then the only possible action is to wait until the second replica is online or until an additional failure has happened, i.e. $A_{1\rightarrow 2} = \{1 \rightarrow 2\}$. Further, in any state, choosing to stay in the same state, is a valid action, i.e. $\forall i \in \mathbf{I_{ON}} \cup \mathbf{I_{OFF}} : i \in A_i$.

The recovery rates can be estimated directly from historical data. The transition probabilities corresponding to failures can be determined in two ways. First, in real deployments the transition rates can be derived directly from observed replica failures, e.g., using maximum likelihood estimation. Second, they can be determined from the PM failure distribution. Given a failure burst, we can compute the expected fraction of copies made unavailable by the burst. Assuming that copies are uniformly distributed across the PMs of the cell, the replica failure rates in the model can be determined from the device failure distribution combinatorically [42]. This approach will be taken in Section 3.5.

Figure 3.1b illustrates that in our model the replication state of each service is observed periodically, namely at the start of each MI n. For each service the first MI ($n = 0$) takes place upon initial deployment. Subsequent MIs ($n > 0$) are separated by the management period h.

At the start of MI n, the current replication state i of the service is observed and the replication manager decides to take action $j \in A_i$. The cost incurred to the system for taking



(a) Illustration of a replication model.

(b) Service request.

Figure 3.1: Problem description.

Parameter	Description
δ	Required lifetime in time units.
T	Required lifetime in MIs.
R	Reward received when available at $n = T$.
V	Penalty when not available at $n = T$.

Table 3.2: Service parameters as per SLA.

action j is given by $C_{i,j}(n)$.

3.3.2 SLA model

The following SLA is considered. A service request has a duration δ, comprising exactly T management periods. Hence, for this service a decision will be made at the start of MI $n \in \{0, \ldots, T-1\}$. If the service is accessible until δ time units after initial deployment, then the service provider receives a reward R. In case the service is no longer available, then the provider receives a penalty V. Figure 3.1b illustrates the flow of a request. Table 3.2 provides an overview of the service parameters.

3.3.3 Formal problem description

The GRMP is formally defined as follows. *Given the replication and service model defined in Section 3.3.1 and Section 3.3.2 respectively, and that revenue of the provider is determined by the costs that he makes and the potential reward or penalty that he receives from the user. In order to maximize expected revenue, which action should the replication manager take at MIs* $n = 0, 1, \ldots, T-1$*?*

3.4 Algorithmic description

Given the inputs described in Table 3.1 and Table 3.2, we maximize the expected reward over the entire request duration. The decision variables are listed in Table 3.3. The structure of this section is the following. First, the algorithmic approach is described. Then, we present the algorithm in pseudo-code.

Parameter	Description
$R_{i,j}(n) \in \mathbb{R}$	The expected reward of taking action $j \in A_i$, when in state $i \in I$ at the start of MI $n \in \{0,\ldots,T-1\}$.
$J_i(n) \in A_i$	The action to take, when in state $i \in I$ at the start of MI $n \in \{0,\ldots,T-1\}$.
$R_i(n) \in \mathbb{R}$	The expected reward corresponding to taking action $j \in A_i$, when in state i at the start of MI $n \in \{0,\ldots,T-1\}$.

Table 3.3: Decision variables for the GRMP.

3.4.1 Approach

The memorylessness-property of the Markov model implies that the behavior of the replicated service at the start of each MI is only determined by its current replication state i, hence the same goes for the expected reward. At the start of MI n, the actions that will be taken for MI $n+1$ up-to $T-1$ are unknown. We denote the best action when in state i at MI n as $J_i(n) \in A_i$.

The expected reward of taking action $j \in A_i$ at MI n, namely $R_{i,j}(n)$, is determined by two elements. First, the immediate cost of this action. Second, the probability to transition from j at n to all other states $\tilde{j} \in \mathbf{I}$ at the start of MI $n+1$, and associated expected reward $R_{\tilde{j}}(n+1)$. Therefore $\forall n \in \{0,1,T-1\}$:

$$R_{i,j}(n) = -C^{(i,j)} + \sum_{\tilde{j} \in \mathbf{I}} P_{j,\tilde{j}}(n) R_{\tilde{j}}(n+1) \tag{3.1}$$

For state $i \in \mathbf{I}, n = T$ the reward is defined by the SLA:

$$R_i(T) = \begin{cases} -V, i \in \mathbf{I}_{OFF} \\ R, i \in \mathbf{I}_{ON} \end{cases} \tag{3.2}$$

Hence, using Equation 3.1 and Equation 3.2 we can recursively determine the actions that maximize expected reward. $\forall n \in \{0,1,T-1\}, i \in \mathbf{I}$:

$$J_i(n) = \arg\max_{j \in A_i} R_{i,j}(n) \tag{3.3}$$

and

$$R_i(n) = \max_{j \in A_i} R_{i,j}(n) \tag{3.4}$$

Summarized, to maximize the overall expected reward at the start of MI n, one needs to select the action with the highest expected reward, given the current replication state $i \in \mathbf{I}$. To calculate the expected reward for each possible action it suffices to know the cost of this action; the transition probabilities to all states in $\mathbf{I}$; and the associated expected rewards at the start of the next MI.

3.4.2 Algorithm

Using the approach described in the previous section, we can find the best policy recursively. Algorithm 3.1 describes how the best strategy can be found using DP. The values of $J_i(n)$ and

$R_i(n)$ will be stored in tables **J** and **R** respectively, to prevent unnecessary recalculation. First, the decision table (**J**) and the table containing the expected values (**R**) are initialized (Line 4 and 5). $J_i(n) == \emptyset$ signifies that the entry has not yet been calculated. `Calc(i,n)` returns the expected reward for (i,n) and stores the calculated values of the decision variables in the appropriate tables. During this function call, all decisions and expected rewards for all states in **I** and for MI $n+1,\ldots,T$ will be either retrieved from the tables, or calculated and stored. If the decision for (i,n) was already tabulated, then the corresponding value is returned (Line 10). If after initial deployment, insufficient replicas are accessible, then the expected value is $-V$ (Line 15). When the service has reached the deadline and sufficient copies remain, then the expected value is R (Line 20).

In all other cases, the best decision and corresponding expected reward must be evaluated for all possible actions (Line 23). Then, the action with the highest expected reward is selected (Line 25) and the maximum expected value is stored and returned (Line 27). Note that, the values are written to the table to prevent unnecessary recalculation of **J** (Lines 14, 19, 25) and **R** (Lines 13, 18, 26).

Algorithm 3.1 Proposed optimal replication algorithm.

```
 1: var I, I_ON, A, C, P, T, R, V
 2: for each i ∈ I do
 3:     for each n ∈ 0, 1, ... T do
 4:         J_i(n) = ∅
 5:         R_i(n) = 0
 6:     end for
 7: end for
 8: procedure CALC(i, n)
 9:     if J_i(n) != ∅ then
10:         return R_i(n)
11:     end if
12:     if i ∉ I_ON && n > 0 then
13:         R_i(n) = −V
14:         J_i(n) = 0
15:         return −V
16:     end if
17:     if N == T && i ∈ I_ON then
18:         R_i(n) = R
19:         J_i(n) = 0
20:         return R
21:     end if
22:     for each j ∈ A_i do
23:         R_{i,j}(n) = −C^{(i,j)} + Σ_{j̃∈I} P_{j,j̃}(n) CALC(j̃, n+1)
24:     end for
25:     J_i(n) = arg max_{j∈A_i} R_{i,j}(n)
26:     R_i(n) = R_{i,J_i(n)}(n)
27:     return R_i(n)
28: end procedure
```

The worst-case complexity can be derived as follows. The decision table contains $|\mathbf{I}|\times(1+T)$ entries. However, for the column at $n=T$ no actions need to be calculated (Line 14 and 19). For each entry (i,n) (Line 23), $|A_i|$ possible actions must be evaluated. We introduce A as the maximum number of actions for any state, i.e. $A=\max_{i\in\mathbf{I}}|A_i|$. Evaluation of each action requires $|\mathbf{I}|$ multiplications and additions. Hence, filling in the table is $\mathcal{O}(|\mathbf{I}|^2TA)$. Note that we

opted for a recursive formulation to ease readability. In a real deployment, overhead should be limited by an iterative implementation.

3.5 Performance evaluation

We implemented a discrete event simulator in Java to simulate a single cell with 100 PMs. Since the model approximates devices within a single cluster to be homogeneous, individual nodal and bandwidth requirements can be ignored, as long as the aggregate capacity within the cluster suffices. Additionally, the number of PMs does not impact the expected reward, as only the PMs on which the service is hosted impact the costs and availability. However, increasing the number of simulated PMs does impact the simulation time as more failure events will be generated. Failures arrive at a rate λ. To generate the number of failures in one failure event we used the bi-exponential model proposed in [44]. In this model, the probability that in a given failure event i out of u PMs fail is given by

$$p_i = (1-\alpha)f(\rho_1, i) + \alpha f(\rho_2, i), \tag{3.5}$$

where α is a tunable parameter that describes the probability of large-scale correlated failures; and $f(\rho, i) : \rho \in \mathbb{R}_{>0}, i \in \{1, \ldots, u\}$ is an exponential distribution for which $f(\rho, i) = c(\rho)\rho^i$. The normalizing factor $c(\rho)$ serves to make $\sum_{i=0}^{u} f(\rho, i) = 1$.

For each PM the Mean Time To Failure (MTTF) is given by

$$MTTF = \frac{u}{\lambda \sum_{i=1}^{u}(ip_i)}. \tag{3.6}$$

Equation 3.6 can be used to generate failures with the same burst size distribution, but with different MTTFs by varying λ. We consider $\alpha = 0.009$, $\rho = 0.3$ and $\rho = 0.96$, unless specified otherwise. These values were extracted from real live deployments on PlanetLab [44]. In the experiments, recoveries can only be initiated at the start of a MI. To prevent accumulation of failure events during an MI, the period must be sufficiently smaller than the MTTF. Under this condition, the exact value of the MI does not influence the expected reward.

3.5.1 Static naive replication

3.5.1.1 Simulation setup

In this setup, we consider a replicated system with L_{min} and L_{max} equal to 1 and 5 respectively. A hosting cost model with a cost per unit time of 0.1 per replica is used. The cost is proportional to the number of duplicates reserved. Two replication algorithms are considered. First, *Optimal time-dependent* is our proposed algorithm. Second, *Fixed Replication(L)* is the default replication strategy in HDFS. This strategy always tries to return to a predefined RL, $L = 1, \ldots, 5$.

3.5.1.2 Results

The influence of the MTTF on the expected reward is shown in Figure 3.2. The request duration is 1000, and $h = 100$. R and V are 1000 and 10000 respectively. The mean instantiation delay

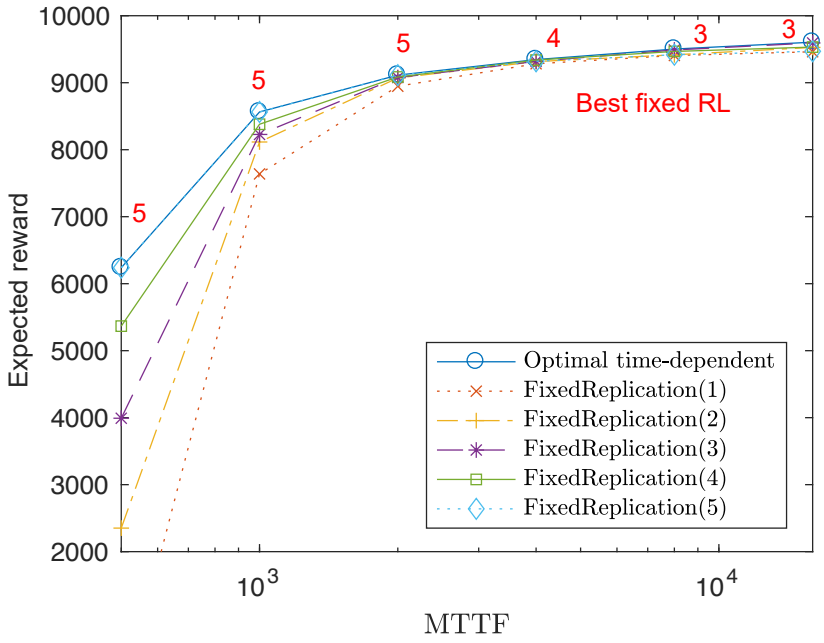


Figure 3.2: Static naive replication: influence of the MTTF on expected reward.

equals 10, for any recovery process. For all MTTFs the expected reward per request of our proposed algorithm is equal to, or higher than for the fixed replication strategies. For low MTTF values, the expected reward is dominated by the probability of data loss. Here, the expected reward per request is equal to that for $FixedReplication(5)$. When the MTTF goes up, then the expected reward increases for all fixed RLs. For high MTTF values, failures are less present and the best policy will be to reduce the operational costs. As the MTTF increases, the fixed RL that yields the highest expected reward decreases from 5 to 3 replicas.

In Figure 3.3, the cost per unit time is varied from 0.1 to 16. Increasing the hosting cost has a dramatic effect on the expected reward. Clearly, our algorithm performs best for all cost levels as it is aware of the operational costs. For low operating costs, the maximum RL is optimal. However, when the cost increases, then the maximum RL performs worse. For a cost per unit time of 0.8, our algorithm performs 17% better than the others. For a cost per unit time of 1.6, our approach is still profitable (+448) while the others have a negative expected reward (-498 and worse). These results show the importance of considering operational costs.

Next, we investigate the influence of request duration on the expected reward in Figure 3.4. $MTTF = 8000$, instantiation delay=10, $h = 100$, cost per unit time=0.1, R=10000, V=100000. Reward and penalty are assumed proportional to the request duration, which is more or less realistic. For low request durations, all six algorithms perform similarly as the probability of failure is negligibly low. When the request duration increases, then the expected reward increases for all algorithms. The performance of $Fixedreplication(5)$ coincides with our algorithm. Note that this is because the cost per unit time is only 0.1. Hence, for the

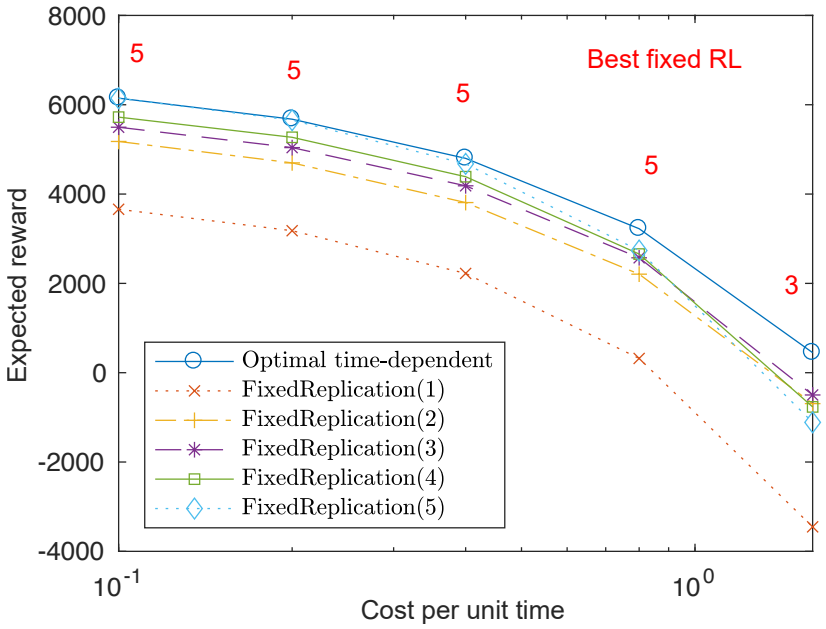


Figure 3.3: Static naive replication: influence of the hosting cost on the expected reward for MTTF=8000.

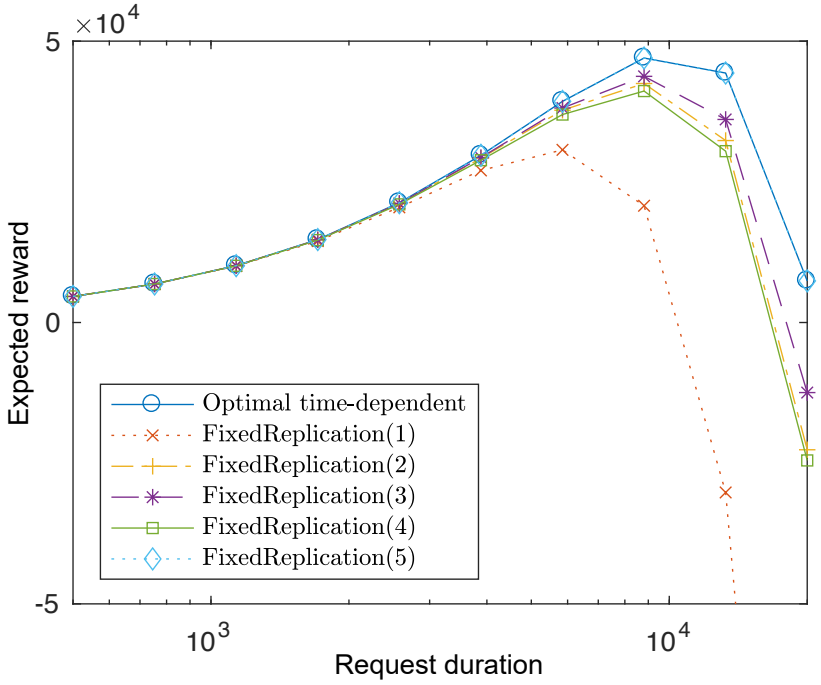


Figure 3.4: Static naive replication: influence of the required lifetime on the expected reward.

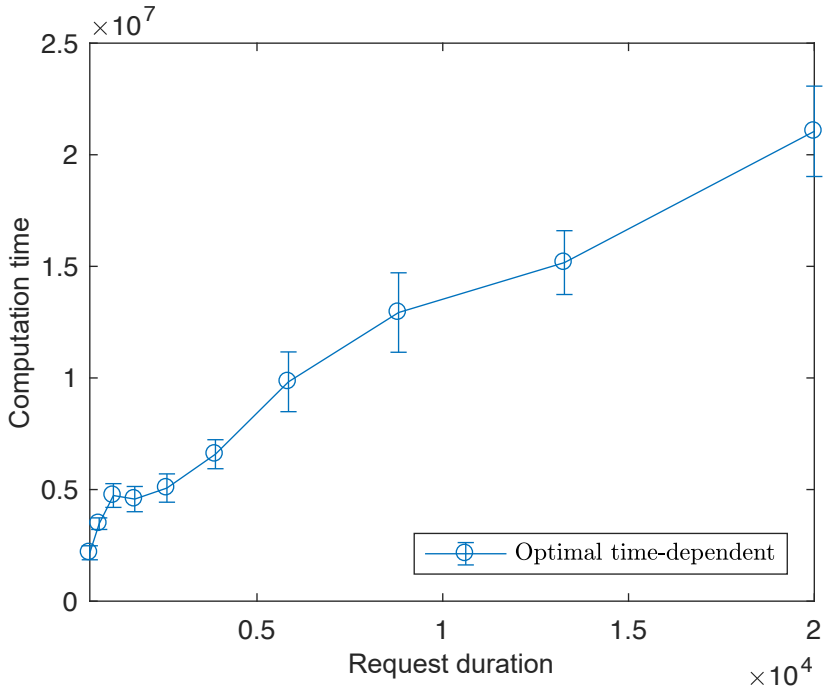


Figure 3.5: Static naive replication: influence of the required lifetime on computation time.

maximum duration of 20000 time units, the total operating cost per request can only vary between 0 and 1000. However, the pay-off can be either 10000 or -100000.

The required computation time to generate the decision table for each request is shown in Figure 3.5. It is averaged out over 10 runs. Since the duration of the MI is fixed, the total number of MIs is proportional to the request duration. Hence, the computation time is proportional to the request duration, as determined in Section 3.4.2.

3.5.2 Static Erasure Coding replication

3.5.2.1 Simulation setup

In this setup, we consider a $(10, 14)$-erasure coded system. This time, as cost we consider the data transferred by the recovery process. In a (k,n)-erasure coded system, the recovery from f failures requires two steps. First, the original data is reconstructed from any combination of k available data blocks. The size of each data block, i.e. the block size, equals b. The stripe size is the total size of the reconstructed data, i.e. $k \times b$. Second, from this reconstructed data, f new data blocks are created and transferred over the network towards their destination. In total, this recovery process requires $(f + k) \times b$ to be transferred. We compare our algorithm to $LazyReplicaton(L)$, for $L = 0, 1, 2, 3$ [43]. In the lazy replication scheme, a recovery is only instantiated when more than L failures have been detected at the start of an MI. Again, the request duration is 1000. $R = 10000$, $V = 100000$, $h = 100$ and $MTTF = 8000$. The mean recovery time is 50.

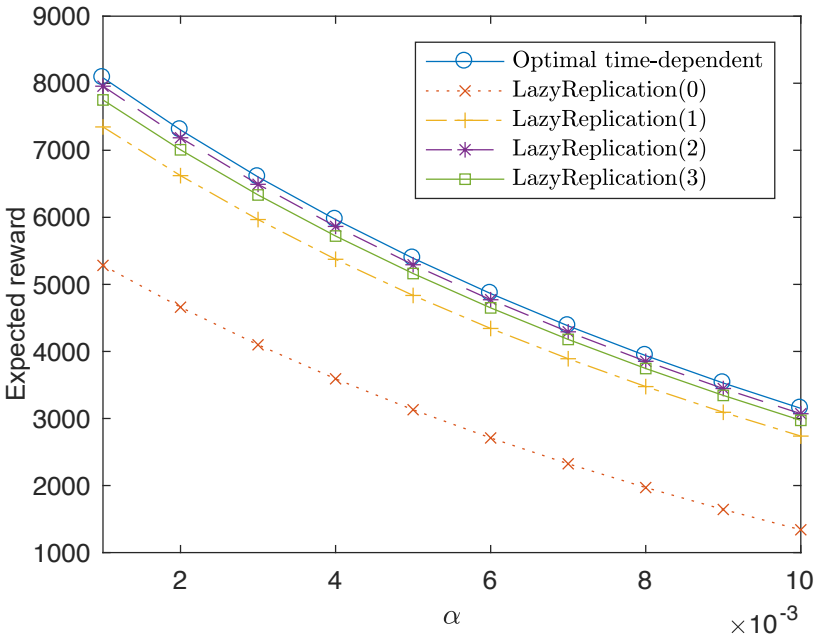


Figure 3.6: Static EC replication: expected reward as a function of the failure correlation (stripe size=256 MB).

3.5.2.2 Results

Figure 3.6 and Figure 3.7 show the impact of the probability of a large-scale failure burst (α), for a stripe size of 256 MB and 512 MB, respectively. For low α, the failures are relatively uncorrelated, while for high α there is a high probability of a large-scale failure burst. Because correlated failure increases the probability of data loss, the expected reward goes down as the failure correlation increases. Overall, our proposed algorithm exceeds the performance of the others. For a stripe size of 256 MB, $LazyReplication(2)$ performs second best, while for a stripe size of 512 MB $LazyReplication(3)$ performs second best. This can be explained by a doubling of the recovery cost, which results in an optimal policy with a higher probability of data loss.

Figure 3.8 and Figure 3.9 show the relation between data loss probability and expected transfer costs for the generated policies. The reward is varied logarithmically between 10^2 and 10^8, and the penalty equals 0. It is clear that the performance of our proposed algorithm dominates the lazy replication algorithm. Moreover, the distribution of data points shows that the proposed algorithm can generate a rich set of replication strategies, that trade-off recovery costs for data loss. This figure illustrates the importance of balancing availability and operational costs to maximize expected revenue.

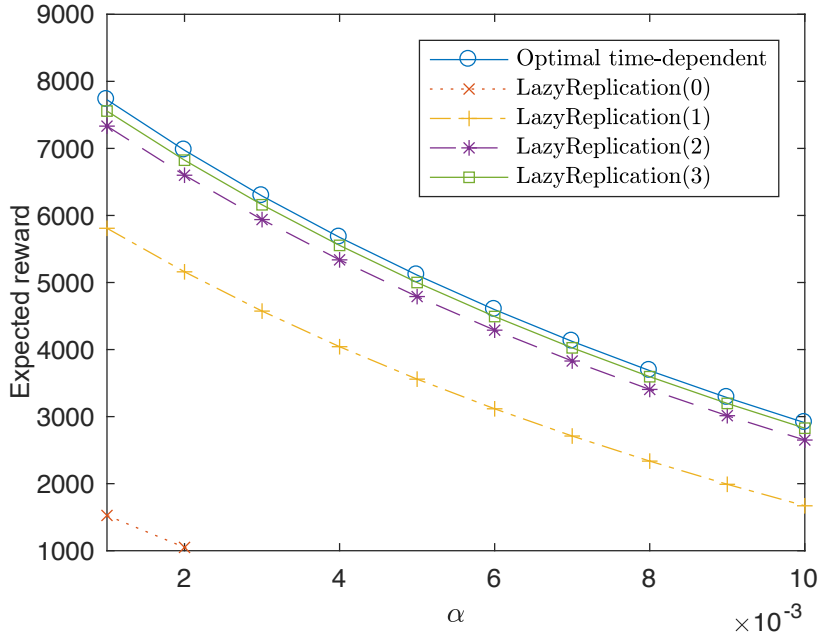


Figure 3.7: Static EC replication: expected reward as a function of the failure correlation (stripe size=512 MB).

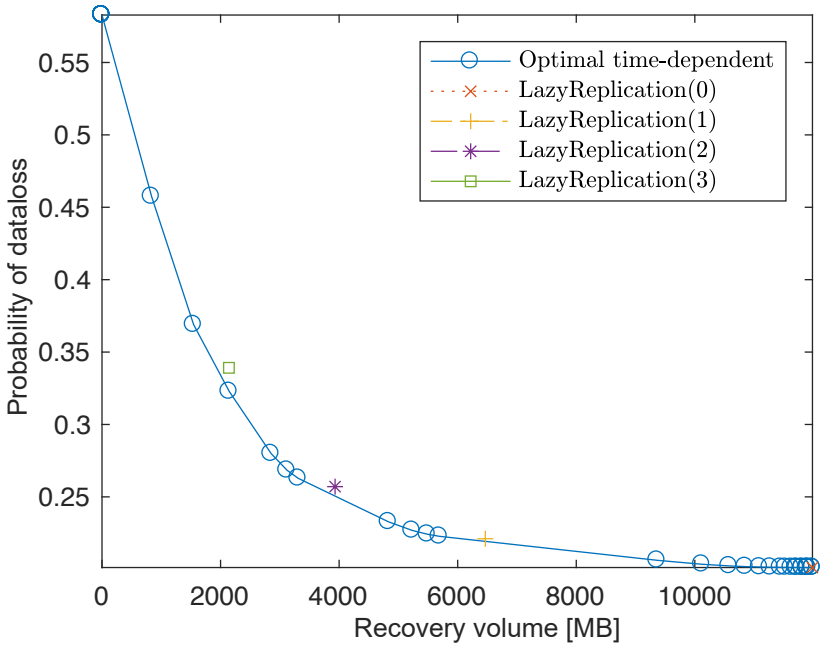


Figure 3.8: Static EC replication: relation between recovery volume and probability of data loss (MTTF=2000).

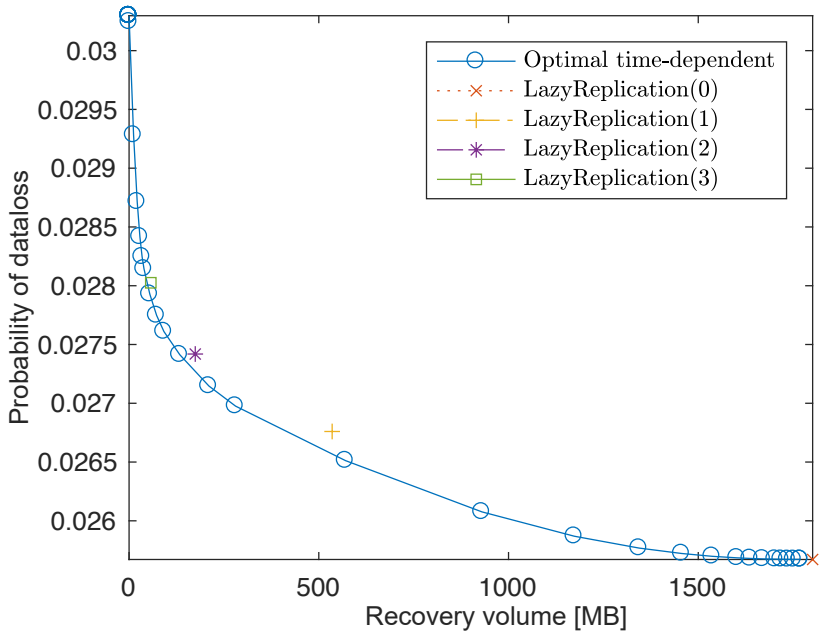


Figure 3.9: Static EC replication: relation between recovery volume and probability of data loss (MTTF=15000).

3.5.3 Dynamic replication

3.5.3.1 Simulation setup

In this setup we consider naive replication with minimum 1 and at most 10 replicas. Again, we consider a hosting cost with a cost per unit time of 0.1. We simulate a dynamic environment, where the failure behavior changes abruptly. $MTTF = 5000$ for $t \in [0, 500000[$ and equals 10000 for $t \in [500000, +\infty[$.

The request duration is increased to 10000, with $h = 100$, with both R and V equal to 100000. The mean time between requests is exponentially distributed with mean 100. In total 10000 service requests are generated.

Every 100000 time units the failure arrival rate λ is re-estimated using a rolling window of 100000 time units. Based on this $\hat{\lambda}$ the failure probabilities are re-estimated and a new decision table is generated. Because the reward and duration of the generated requests are identical, this table can be shared by all services.

3.5.3.2 Results

For each service request the accumulated reward (including penalty, reward and costs) and the accumulated cost is tracked. At the start of each MI the current replication state is evaluated. When the service is offline, then a penalty is incurred. In case the deadline has been reached, then a reward is added. In both cases a reward or penalty is added, and the accumulated reward is written to a log file. No future MIs are scheduled for this service. In case the deadline

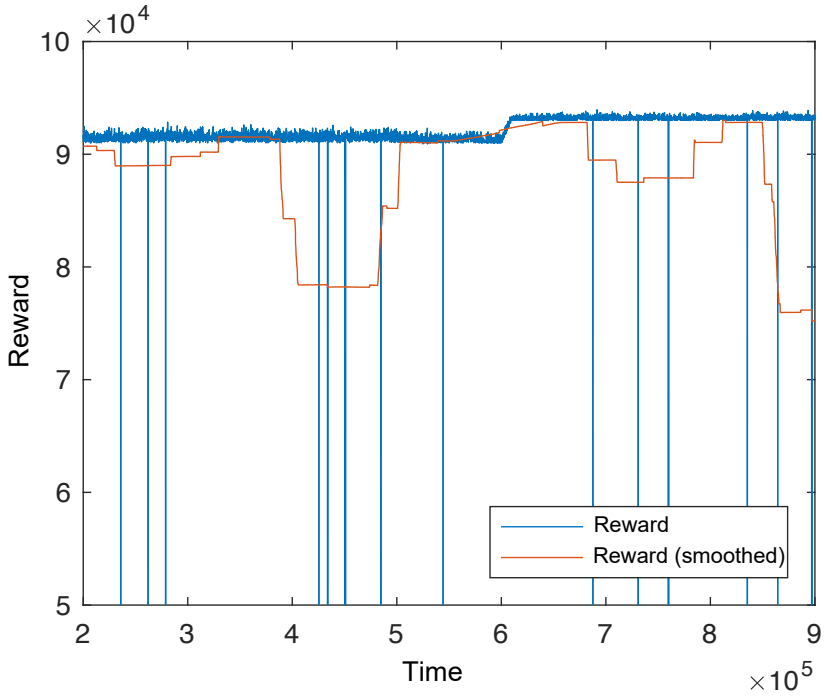


Figure 3.10: Dynamic replication: reward per request.

was not reached (yet), then the replication manager consults the decision table. If the decision table is older than 100000 time units, then a new decision table is generated. The replication manager selects the next action from the table and the corresponding cost is accumulated.

Figure 3.10 and Figure 3.11 shows the received rewards and hosting costs incurred for the generated requests. The smoothed traces result from a moving average with window size 1000. While the failure probability is lowered at $t = 500000$, the decision table is only updated with the new value of $\hat{\lambda}$ at $t = 600000$. In $[500000; 600000[$, the expected reward increases gradually, as the probability of data loss goes down. While this is not immediately clear in the raw data, the smoothed line clearly goes up. In this same period, the average costs increase slightly, because more services live through their entire required lifetime, resulting in a higher overall resource consumption. At $t = 600000$ the decision table is updated. The reward goes up linearly in $[600000; 610000[$, because the decision table is updated for running services which were already running. At $t = 610000$, we see that both average reward and cost remain stable. Compared to $t < 500000$ our replication algorithm has switched to a lower RL, increasing the average reward and lowering operating costs.

3.6 Conclusion and future work

In this work, we present a novel unified approach towards the analysis or replicated systems and the synthesis of a cost-effective replication strategy. We approach the problem of replica

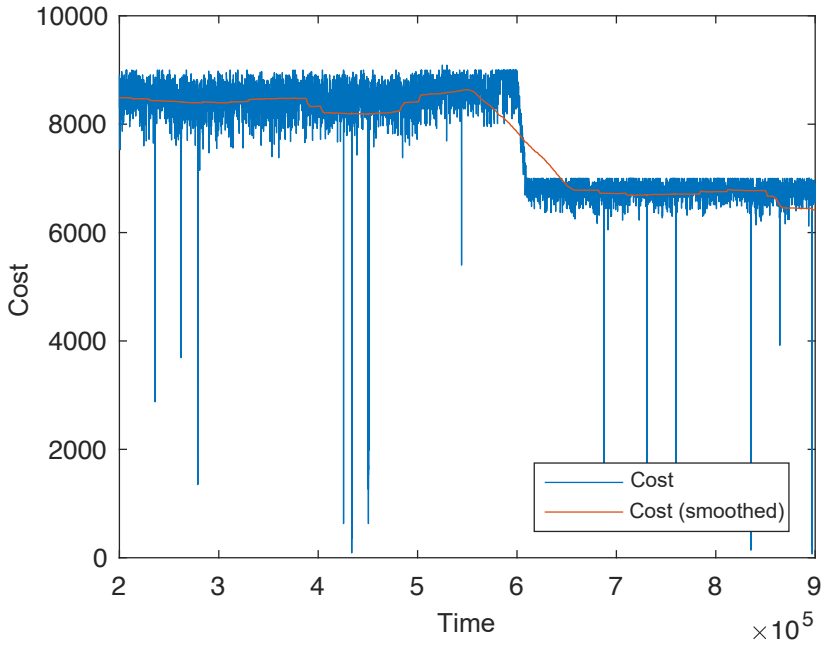


Figure 3.11: Dynamic replication: cost per request.

management in fault-tolerant cloud environments as a runtime revenue optimization problem. While some related works have either solely focused on analysis of replicated systems, others made very unrealistic assumptions on recovery and failure rates. Our approach is based on a dynamic failure model and can be applied to both single datacenters and geo-distributed cloud environments. As the cell dimensions go up, the underlying model becomes more accurate and the computation time remains the same. Therefore, our approach is particularly of interest for revenue optimization in very large-scale cloud networks.

We show that, while other works propose fixed placement strategies for specific cloud environments, our adaptive algorithm yields significant profitability improvements in a wide range of cloud and SLA conditions.

The major limitation of our work is that the computation time of the proposed algorithm is proportional to the number of MIs. Currently, the MI must be small to the MTTF. Therefore, future work includes switching between replication strategies, instead of deciding on individual replication operations at the start of an MI. This improvement can relax the requirement on the MI, in that it only has to be small compared to the required service lifetime.

Chapter 4

Availability-aware application placement

This work was supported by the iFEST and EMD projects. The underlying ideas have been published in [C4, C5, C8, C1].

While the previous chapter focused on the challenges related to the orchestration of persistent resources, this chapter investigates how to orchestrate reliable stateless NSs on top of an unreliable geo-distributed cloud infrastructure. More specifically, it focuses on Question 4, which investigates how to protect NSs against node and link failure in heterogeneous cloud environments.

The strong uptake of cloud computing has led to an important increase of mission-critical applications being placed on cloud environments. Those applications often require high levels of availability coupled with guarantees on a minimum level of throughput and a maximum level of response time. To achieve the lowest response time possible, clouds are more and more decentralized, leading to a heterogeneous network of micro clouds positioned on the edge of the network and possibly interconnected by best-effort PLs. This heterogeneous environment introduces important challenges for the management of these clouds as the heterogeneity results in an increased failure probability. In this chapter, we address these challenges by providing a resilient placement of mission-critical applications on geo-distributed clouds. We present an exact solution to the problem, which is complemented by two heuristics: a near-optimal distributed genetic meta-heuristic and a scalable centralized heuristic based on subgraph isomorphism detection. A detailed performance evaluation shows that, with the newly proposed heuristic based on subgraph isomorphism detection, we can double the amount of applications satisfying availability requirements, in cloud environments comprising over 100 PMs, while keeping the time required to calculate the solution under 20 seconds.

4.1 Introduction

In this chapter, we consider the problem of processing an initial collection of application requests upon startup of a cloud environment, which is referred to as the offline APP [16]. We model the

application requests as VNes, consisting of VNos and their required communication channels. The cloud infrastructure is modeled as a SNe consisting of PMs and their interconnecting PLs. The problem of mapping the VNes to a SNe, whilst considering failures in the SNe is known as the SVNE problem [55]. Given the failure behavior of the cloud infrastructure, we solve the problem of initial distribution of application components over the cloud environment, whilst satisfying a minimum level of total availability for each application.

To the best of our knowledge, this work is the first that provides a computationally feasible approach to place applications in a realistic and large-scale failure prone cloud environment that provides guarantees on the availability. More specifically, the contributions of this work are four-fold. First, we present a novel approach of placing applications in a failure prone cloud environment: by adding additional replicas we can provide availability guarantees. This is formulated as an ILP, which can be used to find an exact solution. Second, we propose a distributed fault-tolerant meta-heuristic based on genetic programming: a distributed set of workers concurrently search for the best placement of applications (including the definition of replicas). Third, we present a scalable centralized algorithm using the paradigm of subgraph isomorphism: this heuristic approach provides ultra-fast placement of applications with a cost in optimality. Fourth, based on an extensive performance evaluation that investigates the performance of different application types, we provide clear guidelines on when and how to apply which application placement algorithm.

The remainder of the chapter is organized as follows. Section 4.2 discusses related works on cloud management algorithms and survivability. In Section 4.3 our model for availability is introduced, and in Section 4.4 the APP is formulated as an ILP, which will be used to find exact solutions. In Section 4.5 a near-optimal distributed genetic meta-heuristic, and a scalable centralized heuristic based on subgraph isomorphism detection are presented. Section 4.6 presents simulations results that evaluate the proposed heuristics. Finally, Section 4.8 concludes the chapter and summarizes the key findings.

4.2 Related work

In this section, the state of the art with regard to the APP in cloud environments is discussed. Early work on application placement merely considers nodal resources, such as CPU and memory capabilities. Deciding whether requests are accepted and where those virtual resources are placed then reduces to an MKP [62]. An MKP is known to be NP-hard and therefore optimal algorithms are hampered by scalability issues. A large body of work has been devoted to finding heuristic solutions. For instance, Xu et al. focus on the multi-objective Virtual Machine Placement (VMP) problem [90]. They propose a GA with fuzzy multi-objective evaluation for efficiently searching the large solution space and conveniently combining possibly conflicting objectives. While Yi et al. propose an evolutionary game theoretic framework for adaptive and stable application deployment in clouds [91]. Other works include NIC capabilities as a dimension in the MKP [92] and assumes an over-provisioned inner-network. While plausible within the boundaries of one datacenter, this condition rarely holds when a combination of multiple clouds or even a wireless environment is considered.

When the application placement not only decides where computational entities are hosted, but also decides on how the communication between those entities is routed in the SNe,

then we speak of *network-aware* APP. Network-aware application placement is closely tied to VNE [93].

An example of a network-aware approach is the work from Moens et al. [94]. It employs a Service Oriented Architecture (SOA), in which applications are constructed as a collection of communicating VNos. This optimal approach performs VNo- and VL-mapping simultaneously. In contrast, other works try to reduce computational complexity by performing those tasks in distinct phases [52, 95].

While the traditional VNE problem assumes that the SNe network remains operational at all times, the SVNE problem does consider failures in the SNe. For instance, Ajtai et al. try and guarantee that a VNe can still be embedded in a physical network, after k SNe components fail. They provide a theoretical framework for fault-tolerant graphs [96]. However in this model, hardware failure can still result in service outage as migrations may be required before normal operation can continue.

Mihailescu et al. try to reduce network interference by placing VMs that communicate frequently, and do not have anti-collocation constraints, on PMs located on the same rack [97]. Additionally, they uphold application availability when dealing with hardware failures by placing redundant VMs on separate server racks. A major shortcoming is that the number of replicas to be placed, and the anti-collocation constraints are user-defined.

Csorba et al. propose a distributed algorithm to deploy replicas of VM images onto PMs that reside in different parts of the network [98]. The objective is to construct balanced and dependable deployment configurations that are resilient. Again, the number of replicas to be placed is assumed predefined.

SiMPLE allocates additional bandwidth resources along multiple disjoint paths in the SNe [56]. This proactive approach assumes splittable flow, i.e. the bandwidth required for a VL can be realized by combining multiple parallel connections between the two end points. The goal of SiMPLE is to minimize the total bandwidth that must be reserved, while still guaranteeing survivability against single PL failures. However, an important drawback is that while the required bandwidth decreases as the number of parallel paths increases, the probability of more than one path failing goes up exponentially, effectively reducing the VL's availability.

Chowdhury et al. propose Dedicated Protection for Virtual Network Embedding (DRONE) [57]. DRONE guarantees VNe survivability against a single PL or PM failure, by creating two VNEs for each request. These two VNEs cannot share any PMs and PLs.

Aforementioned SVNE approaches [96, 97, 98, 56, 57] lack an availability model. When the infrastructure is homogeneous, it might suffice to impose that each VNe has a predefined number of replicas. However, in geo-distributed cloud environments the resulting availability will largely be determined by the exact PC, as moving one VNo from an unreliable PM to a more reliable one can make all the difference. Therefore, geo-distributed cloud environments require SVNE approaches which have a computational model for availability as a function of SNe failure distributions and PC.

The following cloud management algorithms have a model to calculate availability. Jayasinghe et al. model cloud infrastructure as a tree structure with arbitrary depth [40]. Physical hosts on which VMs are hosted are the leaves of this tree, while the ancestors comprise regions and availability zones. The PMs at bottom level are PMs where VMs are hosted. Wang et al.

were the first to provide a mathematical model to estimate the resulting availability from such a tree structure [41]. They calculate the availability of a single VM as the probability that neither the leaf itself, nor any of its ancestors fail. Their work focuses on handling workload variations by a combination of vertical and horizontal scaling of VMs. Horizontal scaling launches or suspends additional VMs, while vertical scaling alters VM dimensions. The total availability is then the probability that at least one of the VMs is available. While their model suffices for traditional clouds, it is ill-suited for a geo-distributed cloud environment as PL failure and bandwidth limitations are disregarded.

In contrast, Yeow et al. define reliability as the probability that critical VNos of a virtual infrastructure remain in operation over all possible failures [99]. They propose an approach in which backup resources are pooled and shared across multiple virtual infrastructures. Their algorithm first determines the required redundancy level and subsequently performs the actual placement. However, decoupling those two operations is only permissible when PL failure can be omitted, and PMs are homogeneous.

In previous work [C5], an availability model for geo-distributed cloud networks was introduced, which considers any combination of PM and PL failures, and supports both VNo and VL replication. The aforementioned model was employed to study the problem of guaranteeing a minimum level of availability for applications. Using an ILP formulation of the problem and an exact solver, an increased placement ratio was demonstrated, compared to naive approaches which lack an availability model. While the ILP solver can find optimal PCs for small-scale networks, its computation time quickly becomes unmanageable when the SNe dimensions increase. In [C8], a first heuristic is presented. This distributed evolutionary algorithm employs a pool model, where execution of computational tasks and storage of the population database (DB) are separated.

Compared to previous work, this work presents the following novelties. First, a fast new algorithm, based on subgraph isomorphism detection, is introduced. In contrast to previous work, this new algorithm is scalable and is the only one that is applicable for real-life large-scale environments. Second, a much more extensive evaluation is provided, considering multiple SN topologies and dimensions, and application types. In comparison to our previous work, next to a flat SN, now also real-world Internet type topologies, generated by a transit-stub model, are studied. Additionally, not only unstructured applications, but also more practical application models, namely 3-Tier and MapReduce, are simulated. Third, we carry out a detailed comparative study to the performance of the presented heuristics, relative to traditional placement algorithms, and homogeneous survivability methods. This study provides clear guidelines about the applicability of each algorithm.

4.3 Resilient cloud placement model

4.3.1 Application requests

This work considers a SOA, which is a way of structuring IT solutions that leverage resources distributed across the network [100]. In a SOA, each application is described as its composition of VNos. Throughout this work, the collected composition of all requested applications will be represented by the instance matrix ($\boldsymbol{I}$).

Symbol	Description
A	Set of requested applications.
S	Set of VNos.
ω_s	CPU requirement of VNo s.
γ_s	Memory requirement of VNo s.
β_{s_1,s_2}	Bandwidth requirement between VNos s_1 and s_2.
$I_{a,s}$	Instantiation of VNo s by application a: 1 i.f.f. instanced.
N	Set of PMs comprising the SNe.
E	Set of PLs (edges) comprising the SNe.
Ω_n	CPU capacity of PM n.
Γ_n	Memory capacity of PM n.
p_n^N	Probability of failure of PM n.
B_e	Bandwidth capacity of PL e.
p_e^E	Probability of failure of PL e.
R_a	Required total availability of application a: lower bound on the probability that at least one of the duplicates for a is available.
δ	Maximum allowed number of duplicates.

Table 4.1: Overview of input variables to the CAPP.

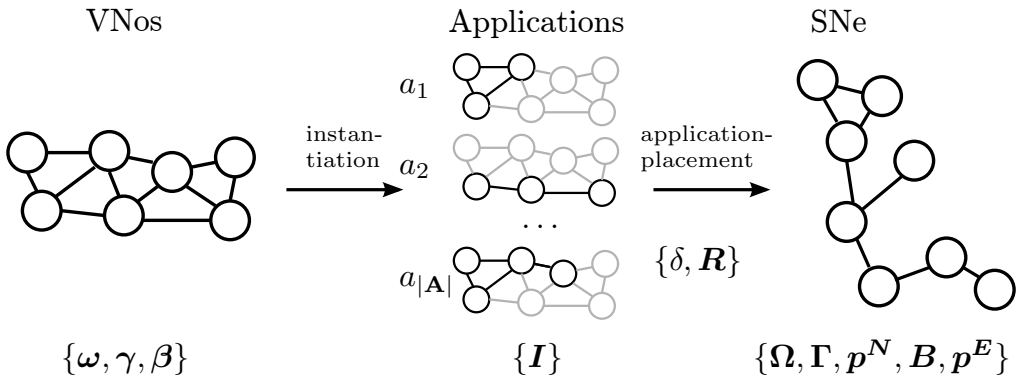


Figure 4.1: Overview of this work: applications $\{\boldsymbol{\omega}, \boldsymbol{\gamma}, \boldsymbol{\beta}\}$, composed of VNos $\{\boldsymbol{I}\}$, are placed on a SNe where PM $\{\boldsymbol{p^N}\}$ and PL failure $\{\boldsymbol{p^E}\}$ are modeled. By increasing the redundancy δ, a minimum availability $\boldsymbol{R}$ can be guaranteed.

VNos have CPU ($\boldsymbol{\omega}$) and memory requirements ($\boldsymbol{\gamma}$). Additionally, bandwidth ($\boldsymbol{\beta}$) is required by the VLs between any two VNos. A submodular approach allows sharing of memory resources among VNos belonging to multiple applications.

	Sharing of resources		
	CPU	**Memory**	**Bandwidth**
Within application	yes	yes	yes
Among applications	no	yes	no

Table 4.2: An overview of resource sharing amongst identical VNos and VLs.

4.3.2 Cloud infrastructure

Consider a SNe consisting of PMs and PLs. PMs have CPU ($\boldsymbol{\Omega}$) and memory capabilities ($\boldsymbol{\Gamma}$). PLs between PMs are characterized by a given bandwidth ($\boldsymbol{B}$). Both PLs and PMs have a known probability of failure, $\boldsymbol{p}^N$ and $\boldsymbol{p}^E$ respectively. Failures are considered to be independent.

4.3.3 The VAR protection method

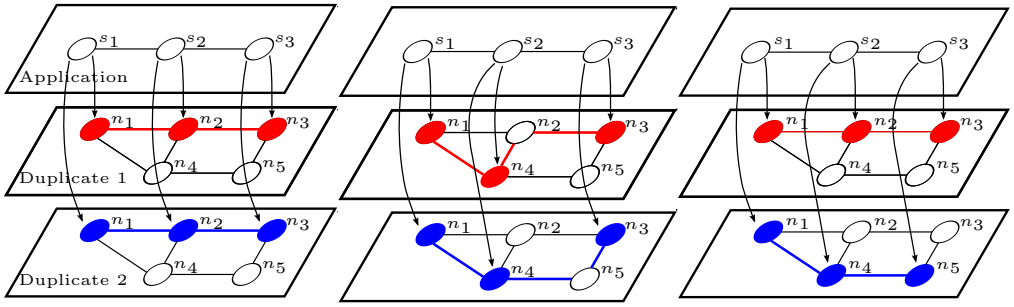
Availability not only depends on failure in the SNe, but also on how the application is placed. Non-redundant application placement assigns each VNo and VL at most once, while its redundant counterpart can place those virtual resources more than once. The survivability method presented in this work, referred to as VAR, guarantees a minimum availability by application level replication, while minimizing the overhead imposed by allocation of those additional resources. VAR uses a static failure model, i.e. availability only depends on the current state of the network. Additionally, it is assumed that upon failure, switching between multiple application instances takes place without any delay. These separate application instances will be referred to as duplicates. Immediate switchover yields a good approximation, when the duration of switchover is small compared to the uptime of individual components. A small switchover time is feasible, given that each backup VNo is preloaded in memory, and CPU and bandwidth resources have been preallocated.

In the VAR model, an application is available if at least one of its duplicates is online. A duplicate is online if none of the PMs and PLs, that contribute its placement, fail. Duplicates of the same application can share physical components. An advantage of this reuse is that a fine-grained trade-off can be made between increased availability and decreased resource consumption. An overview of resource reuse is shown in Table 4.2.

In Figure 4.2 three possible PCs using two duplicates are shown for one application. In Figure 4.2a both duplicates are identical, and no redundancy is introduced. The nodal resource consumption is minimal, as CPU and memory for s_1, s_2, and s_3 are provisioned only once. Additionally, the total bandwidth required for (s_1, s_2), and (s_2, s_3) is only provisioned once. The bandwidth consumption of this configuration might not be minimal, if consolidation of two or three VNos onto one PM is possible. This PC does not provide any fault-tolerance, as failure of either n_1, n_2 or n_3, or $(n_1, n_2), (n_2, n_3)$ results in downtime.

When more than one duplicate is placed and the resulting arrangements of VLs and VNos differ, then the placement is said to introduce redundancy. However, this increased redundancy results in a higher resource consumption. In Figure 4.2b the application survives a singular failure of either (n_4, n_2), (n_2, n_3), (n_4, n_5), or (n_5, n_3). The SVNE depicted in Figure 4.2c survives all singular failures in the SNe, except for a failure of n_1.

Duplicates can be seen as a generalization of the PC model defined by Chowdhury et



(a) 0 Replicated VNos. (b) 1 Replicated VL. (c) 2 Replicated VNos

Figure 4.2: Illustration of the VAR protection method.

al. [57]. The authors place a primary and backup VNe to guarantee survivability against single PM or PL failures, which can be considered two duplicates of the same VNe. There are three key differences between their model and VAR. First, Chowdhury et al. require the placement of exactly two duplicates, while VAR supports any number of duplicates. Second, their approach requires all VNos for one duplicate to be located on different PMs. In our approach, VNos of one duplicate can be consolidated onto one PM. Consolidation offers the possibility to increase availability of a duplicate and avoids wasting precious bandwidth resources. Third, their model does not allow duplicates of the same application to share PMs or PLs. In our model, duplicates of the same application can either have no part of the SNe in common, have some part of the SNe in common (and possibly share resources), or even have completely identical PCs (and require no additional resources). Those three differences mean that their model cannot yield any feasible solution for the problem depicted in Figure 4.2.

4.3.4 Availability calculation

In the previous section, an application was defined available, if at least one of its duplicates is online. Hence, the total availability of an application is then the probability that at least one of its duplicates is available. When at most δ duplicates are considered, then the total availability of application a is given by

$$Z(a) = P\left[\bigcup_{d=1}^{\delta} D_d^a\right] : a \in A, \tag{4.1}$$

where D_d^a denotes the event that duplicate d of application a is available. The event that this duplicate is not available is denoted by $\overline{D_d^a}$.

4.4 Formal problem description

In this section, the problem is formulated as a binary ILP. The input variables to the model were already described throughout Section 4.3. Given those input variables, the algorithm

finds a value for the decision variables listed in Section 4.4.1 that minimizes the objective function (Section 4.4.3). The optimization is subject to the constraints listed in Section 4.4.2.

Symbol	Description
C	Set of physical components in the SNe, i.e. PMs and PLs ($C = N \cup E$).
D	Set of duplicates.
M	Set of minterms.
X	Set of all possible states.
$X(m)$	Particular state of the SNe, the state of each component follows from m according to Equations 4.19 and 4.20.
χ_c	State of physical component c.
$b_c(m)$	Value of χ_c for component c in minterm m.
$\zeta(d,a)$	Availability of duplicate d for application a.
$Z(a)$	Joint availability of application a.

Table 4.3: Overview of auxiliary symbols used throughout the ILP formulation.

4.4.1 Decision variables

The decision variables are described in Table 4.4. O indicates which application requests are accepted, while G provides detailed information about which duplicates are actually placed. Information about the assignment of VNos to PMs is contained in π, Π and U, while υ and Υ tell us how the VLs are routed over the PLs. K, τ and T are directly used for availability calculation. Auxiliary variables are described in Table 4.3.

Symbol	Description
O^a	acceptance of application a: 1 i.f.f. accepted.
$G^{d,a}$	placement of duplicate d of application a: 1 i.f.f. placed.
$\pi_{s,n}^{d,a}$	Placement of VNo s for duplicate d of application a on PM n: 1 i.f.f. hosted.
$\Pi_{s,n}^{a}$	Use of PM n for hosting of VNo s by application a: 1 i.f.f. used.
$U_{s,n}$	Hosting of VNo s on PM n: 1 i.f.f. hosted.
$\upsilon_{s_1,s_2}^{d,a}(e)$	Placement of VL between VNos s_1 and s_2 on PL e for duplicate d of application a: 1 i.f.f. placed.
$\Upsilon_{s_1,s_2}^{a}(e)$	Use of PL e by at least one duplicate of application a for the placement of the VL between s_1 and s_2: 1 i.f.f. placed.
$K_c^{d,a}$	Use of physical component c by duplicate d of application a: 1 i.f.f. used.
$\tau_m^{d,a}$	Coverage of minterm m by duplicate d of application a: 1 i.f.f. covered m.
T_m^a	Availability of application a when the state of the network equals $X(m)$: 1 i.f.f. available.

Table 4.4: Overview of decision variables to the binary ILP.

4.4.2 Constraints

4.4.2.1 Admission control

At most, δ duplicates can be placed for each application:

$$|D| = \delta. \tag{4.2}$$

An application can only be accepted if at least one of its duplicates is placed:

$$\forall a \in A : O^a \leq \sum_{d \in D} G^{d,a}. \tag{4.3}$$

4.4.2.2 Node-embedding

Nodal resources are only assigned to duplicates if they are considered placed:

$$\forall a \in A, s \in S, n \in N, d \in D : \pi_{s,n}^{d,a} \leq G^{d,a} \times I_{a,s}. \tag{4.4}$$

The number of VNos hosted for each accepted duplicate equals the total number of instantiated VNos. If a duplicate is not placed, no VNos are instantiated:

$$\forall a \in A, d \in D : G^{d,a} \times \sum_{s \in S} I_{a,s} = \sum_{s \in S} \sum_{n \in N} \pi_{s,n}^{d,a}. \tag{4.5}$$

If a VNo is hosted on a PM for any of its duplicates, then $\Pi_{s,n}^{a}$ equals 1:

$$\forall a \in A, d \in D, s \in S, n \in N : \pi_{s,n}^{d,a} \leq \Pi_{s,n}^{a}. \tag{4.6}$$

For each duplicate a VNo is hosted on at most one PM:

$$\forall a \in A, d \in D, s \in S : \sum_{n \in N} \pi_{s,n}^{d,a} \leq 1. \tag{4.7}$$

Conservation of CPU and memory resources dictates:

$$\forall n \in N : \sum_{a \in A} \sum_{s \in S} \Pi_{s,n}^{a} \times \omega_s \leq \Omega_n \tag{4.8}$$

and

$$\forall n \in N : \sum_{s \in S} U_{s,n} \times \gamma_s \leq \Gamma_n. \tag{4.9}$$

A VNo must be hosted on a PM, as soon as it is used by one of the duplicates:

$$\forall s \in S, \forall n \in N : \sum_{a \in A} \sum_{d \in D} \pi_{s,n}^{d,a} \leq U_{s,n} \times |D| \times \sum_{a \in A} I_{a,s}. \tag{4.10}$$

4.4.2.3 Link-embedding

MCF constraints on each PM can be expressed as: $\forall a \in A, s_1, s_2 \in S, d \in D, n_1 \in N$:

$$\sum_{(n_1,n_2)\in E} \upsilon^{d,a}_{s_1,s_2}(n_1,n_2) - \sum_{(n_2,n_1)\in E} \upsilon^{d,a}_{s_1,s_2}(n_2,n_1) = \pi^{d,a}_{s_1,n_1} - \pi^{d,a}_{s_2,n_1}. \tag{4.11}$$

$\Upsilon^{a}_{s_1,s_2}(e)$ indicates if at least one of an application's duplicates uses e for this VL: $\forall a \in A, s_1 \in S, s_2 \in S, e \in E, d \in D$:

$$\upsilon^{d,a}_{s_1,s_2}(e) \leq \Upsilon^{a}_{s_1,s_2}(e). \tag{4.12}$$

The total bandwidth used per PL cannot exceed the total PL capacity:

$$\forall e \in E : \sum_{s_1\in S}\sum_{s_2\in S}\sum_{a\in A} \Upsilon^{a}_{s_1,s_2}(e) \times \beta_{s_1,s_2} \leq B_e. \tag{4.13}$$

4.4.2.4 Availability-awareness

For a duplicate to be available, each of the individual components it uses must be available. A component is used by a duplicate if it hosts any of the duplicate's VNos or VLs: $\forall a \in A, d \in D, c \in C, s_1, s_2 \in S$:

$$K^{d,a}_{c} \geq \begin{cases} \pi^{d,a}_{s_1,c} & \text{if } c \in N \\ \Upsilon^{d,a}_{s_1,s_2}(c) & \text{if } c \in E \end{cases}. \tag{4.14}$$

The state of an individual component is described as:

$$\forall c \in C : \chi_c = \begin{cases} 0 & \text{if } c \text{ fails} \\ 1 & \text{if } c \text{ does not fail} \end{cases}. \tag{4.15}$$

The probability that a component fails is given by:

$$\forall c \in C : \mathrm{P}[\chi_c = 0] = \begin{cases} p^{N}_{c} & \text{if } c \in N \\ p^{N}_{e} & \text{if } c \in E \end{cases}. \tag{4.16}$$

The state of the SNe can then be described as:

$$X = (\chi_1, \chi_2, \ldots, \chi_{|C|}). \tag{4.17}$$

To facilitate systematical description of all possible SNe states, which will be further referred to as minterms, the following notation is introduced:

$$M = \{0, 1, \ldots, 2^{|C|} - 1\} \tag{4.18}$$

and

$$\forall m \in M : X(m) = X | \forall c \in C : \chi_c = b_c(m), \tag{4.19}$$

where $b_c(m) \in \{0, 1\}$ is defined by:

$$\forall m \in M : m = \sum_{c\in\{0,1,\ldots,|C|-1\}} b_c(m) \times 2^c. \tag{4.20}$$

As component failures are assumed independent, the probability of each minterm is given by:

$$\forall m \in M : \mathrm{P}[X = X(m)] = \prod_{c \in C | b_c(m)=0} \mathrm{P}[\chi_c = 0] \times \prod_{c \in C | b_c(m)=1} \mathrm{P}[\chi_c = 1]. \quad (4.21)$$

Note that the ILP formulation can easily be extended to support correlated failure, by changing the failure probability in Equation 4.21.

As stated earlier, a duplicate is available, if all physical components that contribute to its placement are online:

$$\forall a \in A, d \in D : \zeta(a,d) = \mathrm{P}\left[\bigcap_{c \in C} (\chi_c{=}1) \cup (K_c^{d,a}{=}0)\right] \quad (4.22)$$

$$= \sum_{m \in M} \tau_m^{d,a} \mathrm{P}[X{=}X(m)], \quad (4.23)$$

where the law of total probability was used:

$$\tau_m^{d,a} = \mathrm{P}\left[\bigcap_{c \in C} (\chi_c{=}1) \cup (K_c^{d,a}{=}0) | X{=}X(m)\right]. \quad (4.24)$$

For the ILP this is reformulated as Equation 4.25. An additional $G^{d,a}$ term ensures that no minterm is covered when a duplicate is not placed ($G^{d,a}{=}0$): $\forall m \in M, d \in D, c \in C, a \in A$:

$$\tau_m^{d,a} \leq G^{d,a} + (b_c(m) - 1) K_c^{d,a}. \quad (4.25)$$

Finally, an application is available if at least one of its duplicates is available:

$$\forall a \in A, m \in M : T_m^a = \mathrm{P}\left[\bigcup_{d \in D} \tau_m^{d,a}\right], \quad (4.26)$$

which can be formulated as:

$$\forall m \in M, a \in A : T_m^a \leq \sum_{d \in D} \tau_m^{d,a}. \quad (4.27)$$

The total availability of an application is then given by:

$$\forall a \in A : Z(a) = \sum_{m \in M} T_m^a \mathrm{P}[X{=}X(m)]. \quad (4.28)$$

Finally, the condition that an application is only placed if the joint availability exceeds R_a can be written as:

$$\forall a \in A : 1 - O^a + \sum_{m \in M} T_m^a \mathrm{P}[X{=}X(m)] \geq R_a. \quad (4.29)$$

4.4.3 Objective function

The placement is sequentially optimized in multiple steps. In each step an objective function is minimized, and the results of the previous steps are added as equality constraints. The objective functions are listed in the order in which they are used by the algorithm.

Maximize acceptance:

$$f_1(A) = -\sum_{a \in A} O^a. \tag{4.30}$$

Minimize bandwidth usage:

$$f_2(A, E, S, \boldsymbol{\beta}) = \sum_{a \in A} \sum_{e \in E} \sum_{s_1, s_2 \in S} \Upsilon^a_{s_1, s_2}(e) \times \beta_{s_1, s_2}. \tag{4.31}$$

Minimize CPU resources usage:

$$f_3(A, N, S, \boldsymbol{\omega}) = \sum_{n \in N} \sum_{a \in A} \sum_{s \in S} \Pi^a_{s,n} \times \omega_s. \tag{4.32}$$

Minimize the number of duplicates used:

$$f_4(A, D) = \sum_{a \in A} \sum_{d \in D} G^{d,a}. \tag{4.33}$$

The last objective function ensures that multiple duplicates of the same application are only placed if beneficial to maximize the placement ratio or minimize resource usage.

4.5 Solution strategies

The ILP formulation presented in the previous chapter can be used to find exact solutions to the problem. In Section 4.6, it will be shown that when the dimensions of the problem increase, even for small instances finding an exact solution to the problem quickly becomes computationally intractable. Therefore, two heuristic algorithms, which can find "good-enough" solutions within a reasonable time-frame, were developed.

4.5.1 GRECO: Genetic Reliable ClOuds

A scalable algorithm to search for a good placement solution, by using a distributed GA, is defined. This algorithm is validated in a pool-based framework, which allows a completely decentralized computation of the solution and can even survive when the PMs that calculate the solution fail. In this section, Genetic Reliable ClOuds (GRECO) is described. Firstly, Section 4.5.1.1 explains the foundations of a GA. Secondly, the chromosome and its decoding, are explained in 4.5.1.2 and 4.5.1.3, respectively. Finally, Section 4.5.1.4 describes the pool model.

4.5.1.1 Genetic Algorithm

GAs are frequently used to solve hard optimization problems. A GA uses a chromosome representation to represent solutions in the solution space. A chromosome representation of one particular solution is referred to as an individual. In a population-based GA, multiple individuals are maintained at each time during execution of the algorithm. The GA starts by creating a random set of individuals, referred to as the seed population. Additionally, through several iterations, the GA selects the best individuals (based on a selection operator). New

solutions are generated in each iteration by combining chromosomes two by two, producing (hopefully better) children. With a small probability there can be a small mutation on one of the chromosomes. The last iteration step is to check the end-condition, which is highly problem-specific. The key idea behind a GA is the use of the laws of natural selection and survival of the fittest to generate the solution [101].

4.5.1.2 Chromosome definition

The key challenge in creating a GA is finding a suitable chromosome representation, because this representation will largely determine its performance. A biased-random-key chromosome is proposed, which is an array of floating-point values between 0 and 1, and is used to make decisions in the decoding phase of the chromosome [102]. A biased-random-key allows definition of a decoder, which can't possibly create invalid solutions. Hence, one can be sure that the offspring of two valid parent solutions, will also be valid solutions.

$$
\begin{aligned}
C = \Big[\; & \underbrace{A_1, A_2, \cdots, A_{|A|}}_{\text{Order of the applications}} \; , \\
& \underbrace{S_1^{1,1}, S_2^{1,1}, \cdots, S_{|S_1|}^{1,1}, S_1^{1,2}, \cdots, S_{|S_1|}^{1,\delta_1}, \cdots, S_{|S_{|A|}|}^{|A|,\delta_{|A|}}}_{\text{Order of the VNos for each duplicate}}, \\
& \underbrace{N_1^{1,1}, N_2^{1,1}, \cdots, N_{|S_1|}^{1,1}, N_1^{1,2}, \cdots, N_{|S_1|}^{1,\delta_1}, \cdots, N_{|S_{|A|}|}^{|A|,\delta_{|A|}}}_{\text{PM number for each VNo in each duplicate}} \Big]
\end{aligned}
\tag{4.34}
$$

The chromosome described in Equation 4.34 consists of three main parts:

1. $A_1, \cdots, A_{|A|}$, describes the order in which the applications are placed.
2. $S_1^{1,1}, \cdots, S_{|S_{|A|}|}^{|A|,\delta_{|A|}}$ describes the order in which the VNos are placed within each duplicate.
3. $N_1^{1,1}, \cdots, N_{|S_{|A|}|}^{|A|,\delta_{|A|}}$ determine which PM hosts which VNo for which duplicate.

4.5.1.3 Deterministic Decoding

When using a biased-random-key, there is no straight-forward way to interpret the chromosome. Therefore, a decoding algorithm is used to translate the chromosome into a solution in the solution space.

Within GRECO, the approach shown in Algorithm 4.1, is taken. First, the applications are ordered by the value of the chromosome's first part ($A_1, \cdots, A_{|A|}$), as this will define the order in which applications are prioritized in the actual placement, the first step of Algorithm 4.1. Placement of an application starts by allocating its first duplicate. If that duplicate is allocated, then the realized availability is calculated (Line 6). If the realized availability exceeds the requested availability, then the application has successfully been allocated, and the decoder proceeds to the next application. On the other hand, if the realized availability is lower than the requested availability, and the maximum number of duplicates for this application has

not been exceeded, then the decoder tries to place an additional duplicate. If the required availability level is not met and the maximum number of duplicates has been placed, then the application request is declined, and its placement removed.

To place duplicate d of application a, first its VNos are ordered by the value of the second part of the chromosome $S_1^{a,d}, S_2^{a,d}, \cdots S_{|S_a|}^{a,d}$, (Line 8). For each VNo s, a list (L) is created, containing only the PMs that can run VNo s, while satisfying all necessary constraints (Line 10). For each PM, it must be verified if the remaining CPU and memory capacity suffice, and if there is sufficient bandwidth to the PMs, that were added to the PC previously. Only PMs that satisfy those three constraints are included in L. If list L is empty, then application a is removed from the placement because there is no valid way to place a (Line 12). Else, if $|L| > 0$, then L is ordered, and the n^{th} PM is selected (Line 15). This PM will host VNo s for duplicate d and the required bandwidth for the VLs between s and the previously handled VNos is allocated. The VLs is routed over the ShP with sufficient remaining bandwidth.

It is mandatory that a certain genome maps to one and only one PC. Therefore, the PMs must have a total deterministic order, independent of the use of the PMs. The PMs are sorted by id, this is easy, deterministic and generally results in a good random order for the last step. The decoding algorithm is illustrated in Algorithm 4.1.

Algorithm 4.1 Decoding algorithm

```
 1: procedure DECODING(A_1.., S_1^{1,1}.., N_1^{1,1}..)
 2:     SORTBYCHROMOSOME(A, A_1..A_{|A|})
 3:     for each a ∈ A do
 4:         δ = 0
 5:         r = 0
 6:         while d ≤ δ_a ∧ r < R_a do
 7:             d += 1
 8:             SORTBYCHROMOSOME(S_a, S_1^{a,d}..S_{|S_a|}^{a,d})
 9:             for each s ∈ S_a do
10:                 L = ALLPOSSIBLEPM(s)
11:                 if |L| == 0 then
12:                     break
13:                 end if
14:                 SORTBYID(L)
15:                 n = ⌈N_s^{a,d} |L|⌉
16:                 PLACEVNO(s, n)
17:             end for
18:             r = CALCULATEAVAILABILITY(a)
19:         end while
20:         if r < R_a then REMOVEAPPLICATION(a)
21:         end if
22:     end for
23: end procedure
```

4.5.1.4 Framework: Pool model

Various models exist to distribute the population among the computing nodes. Amongst those distribution models for population-distributed GAs, the pool model offers the best combination of scalability and fault-tolerance. While in other distribution models, worker nodes (called workers) are responsible of both executing GA operations, and storing part of the population

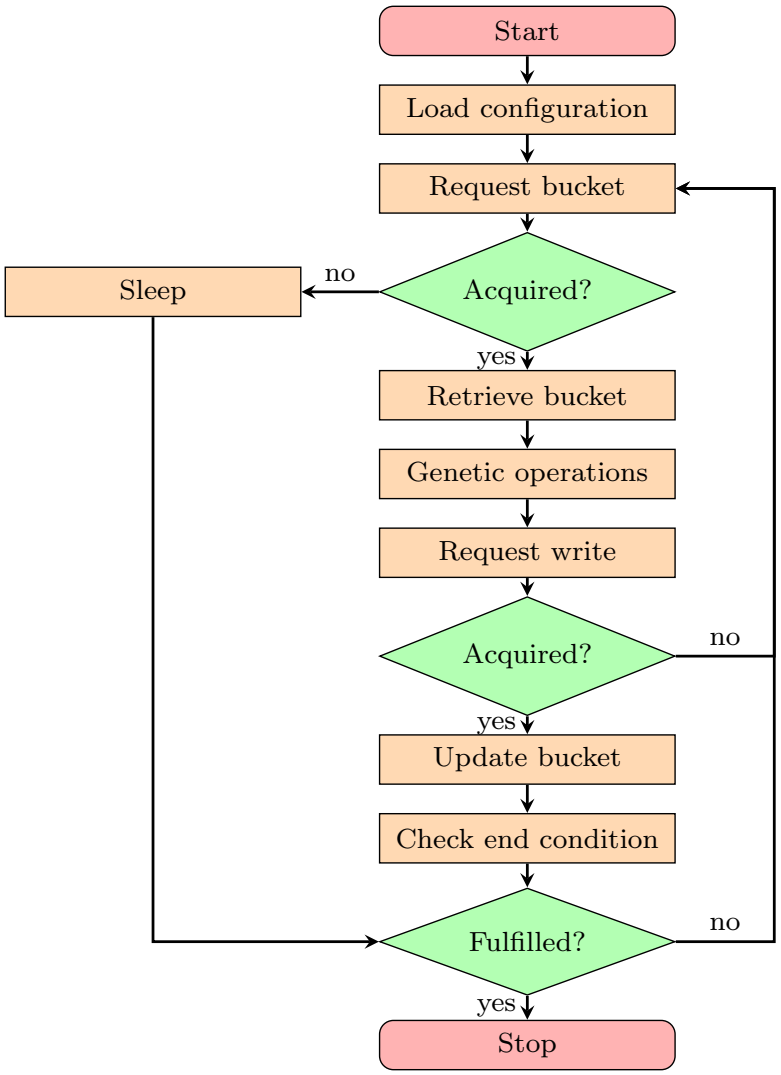


Figure 4.3: Workflow of a GRECO worker

database, the pool model maintains the population in a separate DB. A pool model deploys a set of autonomous processors working on a shared resource pool. The processors are loosely coupled, do not know of each other's existence and interact with only the pool.

The workflow of a GRECO worker is shown in Figure 4.3. First, the worker loads a configuration file from the DB, which describes the application requests, and the SNe. Subsequently, the worker contacts the DB and requests a bucket, which holds a random partition of the population. If a bucket cannot be acquired, because all buckets have already been assigned, then the worker sleeps, and checks the end-condition of the GA. If a bucket can be acquired, then the worker retrieves it and performs the evolutionary operators (selection, crossover, mutation, elite conservation). Next, the worker requests permission to update the bucket in the

Variable	Definition
G^V	Graph representing application requests, comprising VNos and VLs that need to be placed.
G^V_{sub}	Subgraph of G^V, containing the VNos and VLs that have been added to the placement.
G^P	The graph, representing the SNe, comprising PMs and PLs
C	List of (VNo-PM) pairs.
$M(\cdot)$	Mapping of its argument to G^P.
$\mathrm{M}(G^V_{max})$	Mapping for which most applications are placed.
w	Total number of mappings that have been tried.
W	Upper-limit to the number of mappings.

Table 4.5: An overview of the variables used in the vnmFlibm routine.

DB. When the worker cannot update the bucket, e.g., because it took too long to process it, then the worker will try to continue processing other buckets. If the worker gets write permission then it proceeds to update the bucket and checks the end-condition. If the end-condition is met, e.g., the desired number of generations is exceeded, then there is no more work to do and the worker stops, else the worker proceeds to request new work.

4.5.2 Subgraph isomorphism

The APP algorithm presented in this section is based on subgraph isomorphism detection: it maps VNos and VLs during the same stage.

Lischka et al. show that this method results in better mappings and is faster than the two-stage approach, especially for large VNes with high resource consumption which are hard to map [53]. The advantage of this single stage approach is that link mapping constraints are taken into account at each step of the mapping. When a bad mapping decision is detected it can be revised by simply backtracking to the last valid mapping decision, whereas the two-stage approach has to remap all VLs, which is very expensive in terms of runtime. vnmFlib, the backtracking algorithm presented by Lischka et al. allows the mapping of VLs to PPs shorter than a predefined distance value ϵ (in terms of hops).

A modified version of vnmFlib, referred to as vnmFlibm, is described in Algorithm 4.2.

An overview of the variables used in vnmFlibm is shown in Table 4.5.

The algorithm tries to build a valid SVNE solution by successively adding PMs and PLs of G^V to an initially empty subgraph G^V_{sub} of G^V. During the mapping process the algorithm ensures that $M(G^V_{sub})$ is a valid mapping of G^V_{sub} onto G^P. In each step of the algorithm, a list of possible (VNo, PM)-pairs is generated by the function *Genneigh* (Line 5). The pairs are ordered by *Genneigh* as follows. First, the (VNo, PM)-pairs are ordered application-wise, which assures that at most one partially placed application is present in $M(G^V_{sub})$. The application order is determined subsequently by total increasing CPU requirements, memory requirements and finally by application number. Second, the (VNo, PM)-pairs are ordered by VNo index. Third, they are ordered by duplicate number. Finally, they are ordered by the id of the PM.

The algorithm loops over this sorted list, and when a valid mapping is encountered, a

new subgraph G^V_{sub} and a corresponding mapping are generated (Line 9). The validity of the mapping is checked by the *valid* function. The *valid* function (Line 7) checks if PM n^P has sufficient remaining nodal resources to host VNo n^V, and if sufficient bandwidth remains in the SNe to route all VLs to and from the VNos already included in $M(G^V_{sub})$. The VLs are routed using a ShP algorithm.

Additionally, the *valid* function verifies if the availability requirement can be met. During availability calculation, the placement of an application is most of the time incomplete. The availability of those unplaced components is assumed 100%. If the potential availability increments, brought by future placement of an additional replica were not taken into account, then the algorithm would not be able to place applications which require multiple duplicates to reach their availability goal. The (intermediate) availability of a mapping will be referred to as $f(M(G^V_{sub}))$ in the example at the end of this section.

The algorithm can terminate in one of two ways. On the one hand, if the maximum number of mappings W has been exceeded (Line 3), or if all (VNo, PM)-pairs have been exhausted (Line 22), then the algorithm returns null. In the calling procedure, the best intermediate result can be used, which was stored in $M(G^V_{max})$. $M(G^V_{max})$ is updated each time a placement is found which holds a higher number of completely placed applications (Line 11). On the other hand, when G^V_{sub} fully covers G^V (Line 18), the algorithm returns $M(G^V_{sub})$ (Line 19), which is a valid mapping of G^V on G^P.

Algorithm 4.2 vnmFlibm procedure.

```
 1: procedure VNMFLIB(G^V_sub, M(G^V_sub), G^V, G^P, M(G^V_sub), M(G^V_max), w, W)
 2:     if w ≥ W then
 3:         return null
 4:     end if
 5:     C← GENNEIGH(G^V_sub, G^V, G^P)
 6:     for each (n^V, n^P) ∈ C do
 7:         if VALID(M(G^V_sub), (n^V, n^P), G^V) then
 8:             w ← w + 1
 9:             create G^V_sub and M(G^V_sub) by adding (n^V, n^P)
10:             if PLACED(M(G^V_sub)) >PLACED(M(G^V_max)) then
11:                 M(G^V_max) ← M(G^V_sub)
12:             end if
13:             M_T ← VNMFLIB(G^V_sub, M(G^V_sub), G^V, G^P)
14:             if M_T !=null then
15:                 return M_T
16:             end if
17:         end if
18:         if G^V_sub == G^V then
19:             return M(G^V_sub)
20:         end if
21:     end for
22:     return null
23: end procedure
```

The algorithm's operation is illustrated using the problem shown in Figure 4.2. An availability A, equal to 0.9853, is assumed for each SNe component. For the sake of clarity, the bandwidth in the SNe is considered not to be a critical constraint, this condition holds if each VL requires 1 unit of bandwidth, and each PL has a bandwidth capability of at least 1 unit. The required availability is 97%. It is assumed that the memory constraints limit the number of

VNos hosted per PM to 1. In Figure 4.4, the different steps of the algorithm are shown. Note that only the mapped (VNo, PM) pairs are explicitly indicated, to ease notation. The routing is performed using a ShP algorithm. For the availability calculation, these paths must be taken into account. Distinction is made between the VNos of the two duplicates, by adding a tilde to the VNos corresponding to the second duplicate.

The inputs to vnmFlibm take following values: $G^V_{sub} = (\emptyset, \emptyset)$, $G_V = (V^V, E^V)$, where $V^V = \{s_1, \tilde{s}_1, s_2, \tilde{s}_2, s_3, \tilde{s}_3\}$, and $E^V = \{(s_1, s_2), (\tilde{s}_1, \tilde{s}_2), (s_2, s_3), (\tilde{s}_2, \tilde{s}_3))\}$. $G^P = (V^P, E^P)$, where $V^P = \{n_1, n_2, n_3, n_4, n_5\}$, and $E^P = \{(n_1, n_2), (n_1, n_4), (n_2, n_3), (n_2, n_4), (n_3, n_5), (n_4, n_5)\}$, $M(G^V_{sub}) = \emptyset$, $M(G^V_{max}) = \emptyset$, $w = 1$, $W = 4 \times 6 = 24$.

Now, the execution commences. In step 1, G^V_{sub} is empty and the algorithm tries to place the first VNo, namely s_1, on n_1, which is a valid placement. In step 2, (s_1, n_1) is added to the mapping, and the algorithm attempts to place $\tilde{s}_1$ on n_1, which is also a valid placement. In step 3, $c = (s_2, n_1)$ is not valid because n_1 does not have enough resources. Therefore, the first valid option is (s_2, n_2). In step 4, $c = (\tilde{s}_2, n_1)$ is not valid because n_2 does not have enough resources. $c = (\tilde{s}_2, n_2)$ is not valid because $f(M(G^V_{sub}))$ would become $A^3 = 0.9565$, which is lower than 97%. n_3 is a valid option. In step 5, n_1 through n_3 do not have enough resources to host s_3. The availability when using either s_4, or s_5 would be too low, as $f(M(G^V_{sub}))$ would be respectively $A^5 + A^4 - A^7 = 0.9696$, and $A^6 + A^4 - A^7 = 0.9559$. Hence, there is no valid (VNo, PM)-pair in C, and the algorithm backtracks (rip up $(\tilde{s}_2, n_3)$). In step 6, the algorithm continues where it left off in step 4, and tries (s_2, n_4), which is a valid combination. In step 7, n_1 and n_2 do not have sufficient resources to host s_3, but n_3 does. In step 8, n_1 and n_2 cannot host $\tilde{s}_3$. n_3 is also not a valid candidate, as the resulting $f(M(G^V_{sub}))$ would be $A^5 + A^6 - A^7 = 0.9421$. n_4 is also not a valid option, because it does not have sufficient remaining resources. n_5 is a valid option. In step 9, $M(G^V_{sub})$ contains one fully placed application, therefore $M(G^V_{max})$ is overwritten. G^V equals G^V_{sub} and the algorithm terminates. The resulting placement is the one depicted in Figure 4.2c.

4.6 Performance evaluation

In this section, the performance of the proposed heuristics is compared against the ILP formulation presented in Section 4.4, and two other placement methods found in literature. For this evaluation, a custom simulation platform is developed in Java. First, this software platform simulates the selected type of application requests and SNe configurations. The result is a json file per problem description, holding the input parameters shown in Table 4.1. Subsequently, the selected application placement algorithm is applied to these input json files and a PC is generated and stored in an output json file. These output files are then further analyzed to compile the graphs shown throughout this section.

This Section is structured as follows. First, the evaluated algorithms are discussed. Second, the models for workload and cloud environments are presented. Then, the effect of multiple parameters is analyzed. For each input parameter, the types of simulated workload and cloud environment are described, and an analysis is provided.

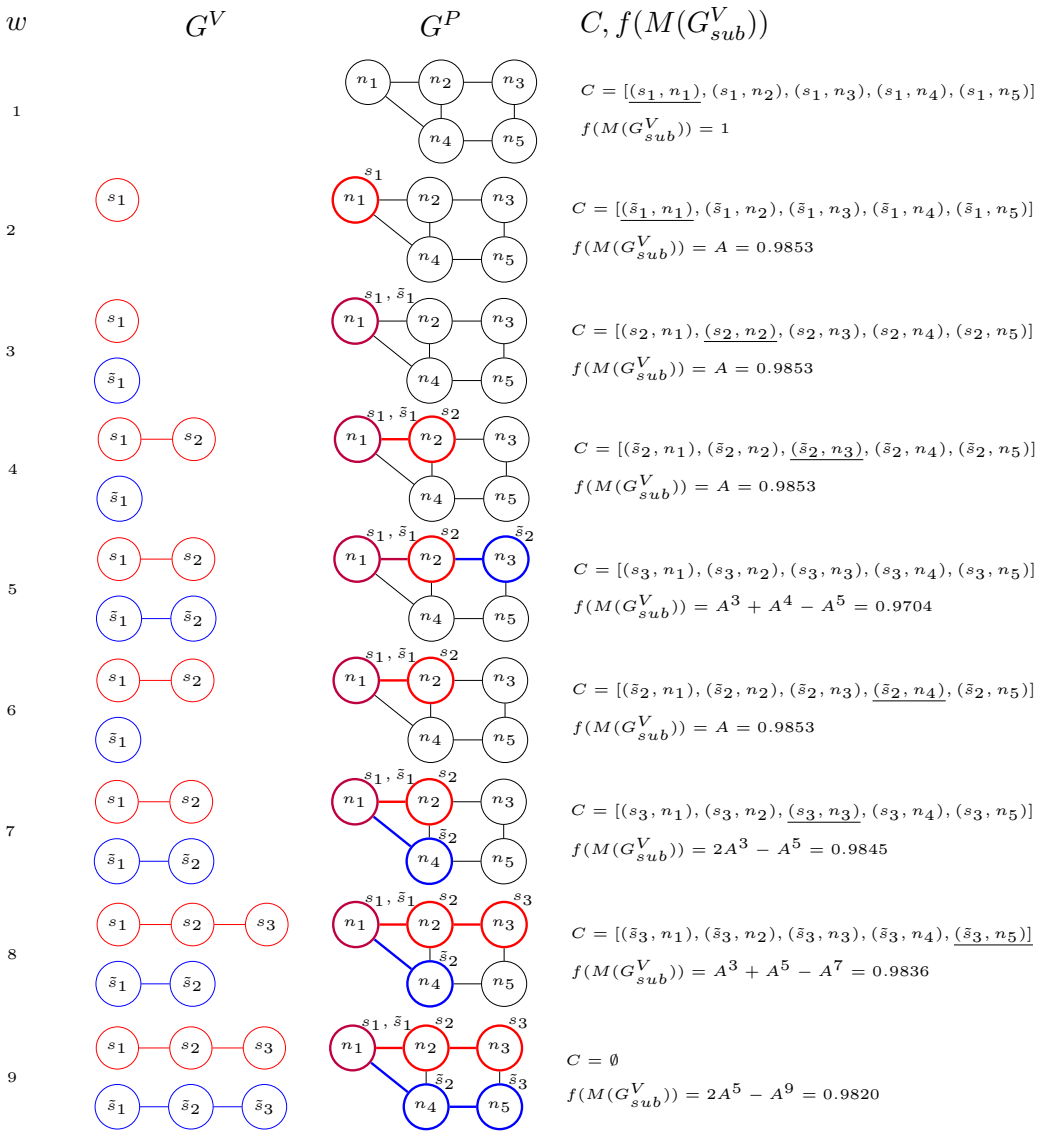


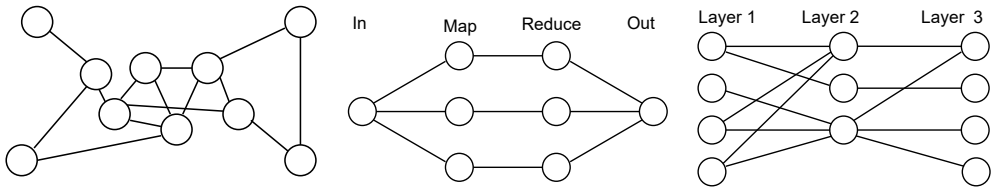
Figure 4.4: Steps taken by vnmFlibm when solving the problem depicted in Figure 4.2.

Method	Duplicates per accepted application	Consolidation onto one PM	Availability
VAR (this work)	at least 1, and at most δ	any two VNos can be consolidated	checked before each mapping
MOENS [94]	exactly 1	any two VNos can be consolidated	checked after execution has finished
DRONE [57]	exactly 2	only VNos of different applications can be consolidated	checked after execution has finished

Table 4.6: Overview of the evaluated placement methods.

4.6.1 Evaluated algorithms

In this section, the suffixes OPT, SUB, and GA indicate an exact algorithm based on an ILP-formulation, a heuristic based on subgraph isomorphism detection, and a GA respectively. An overview of the compared placement methods is given in Table 4.6. VAR refers to the protection method described in Section 4.3.3, which can vary the amount of redundancy introduced. The ILP-formulation presented in Section 4.4, is referred to as VAR-OPT, the GRECO algorithm (Section 4.5.1.1) as VAR-GA, and the heuristic based on subgraph isomorphism detection (4.5.2) as VAR-SUB. To the best of our knowledge, there exist no other placement algorithms, introducing a model for availability which considers both PM and PL failures. Therefore, a comparison is made to methods which do not have a model for availability. After these placement algorithms have finished execution, the applications whose availability requirements have not been met, are removed. Two other placement methods are considered. On the one hand Moens et al. place each application at most once [94]. MOENS-OPT is an exact algorithm based on ILP. MOENS-SUB is a self-defined heuristic based on subgraph isomorphism detection for MOENS-OPT. The only differences between MOENS-SUB and VAR-SUB, are that MOENS-SUB does not have an availability requirement and that it places at most 1 duplicate. On the other hand, DRONE-OPT is based on the ILP-formulation from Chowdhury et al [57]. In their approach, each accepted application request must be placed exactly twice and VNos belonging to the same application cannot be consolidated onto one PM. The embeddings of the primary and backup VNe must be fully disjoint (i.e. they cannot have any PM or PL in common), which guarantees survivability against a single PM or PL failure. For the sake of comparison, their model is modified so that memory resources can be shared among applications which use the same VNo. DRONE-SUB is a self-defined algorithm based on subgraph isomorphism detection for DRONE-OPT. Compared to VAR-SUB, this algorithm does not check availability during placement and it always tries to place two duplicates. Additionally, DRONE-SUB allows an application to make use of each PM and PL at most once, while VAR-SUB has no such restrictions.



(a) Random (b) MapReduce (c) 3 Tier

Figure 4.5: An illustration of multiple application models.

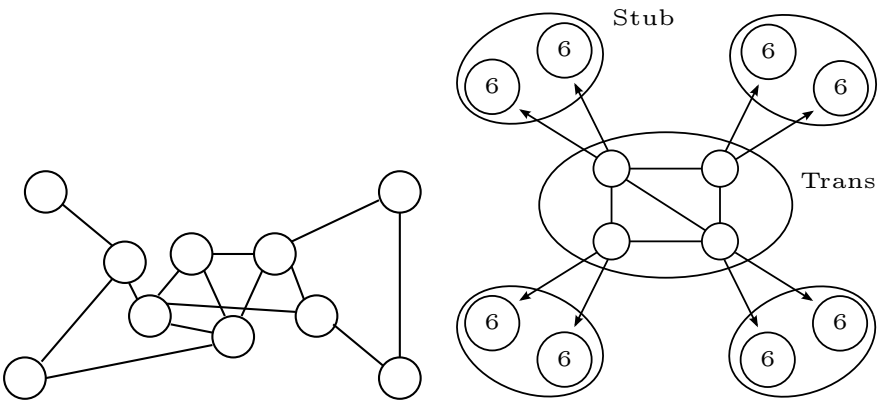
4.6.2 Application model

The SOA model described in Section 4.3.1 can be used to represent a wide range of applications. In the following, three application types are presented: one with a flat hierarchy, and two with a special structure, which are commonly used in software engineering. These application types are used throughout the performance evaluation.

Random This application type is similar to the simulation setup used by Moens et al. [94]. In this model, application requests are generated as follows. First, a certain number of VNos is generated, which have a certain probability to be interconnected pairwise by a VL. Then, each application randomly selects VNos. In this model, multiple applications can comprise the same VNo. The application model is illustrated in Figure 4.5a.

MapReduce MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key [103]. The application model shown in Figure 4.5b comprises 1 mapper and 1 reducer stage and assumes the input and output files to be located on one input and one output node, respectively. It is assumed that resources cannot be reused between VNos of multiple applications. Each mapper communicates with the input node and exactly one reducer node. The reducer nodes communicate with the output node and exactly one mapper node. When there are m mappers in the mapper stage, then there are also m reducers, which totals $2+2\times m$ VNos per application.

3 Tier A multi-tier architecture is a software architecture in which different software components, organized in tiers (layers), provide dedicated functionality. The most common occurrence of a multi-tier architecture is a three-tier system consisting of a data management tier (mostly encompassing one or several database servers); an application tier (business logic); and a client tier (interface functionality). Conceptually, a multi-tier architecture results from a repeated application of the client/server paradigm. A component in one of the middle tiers is client to the next lower tier and at the same time acts as server to the next higher tier [104]. The simulations assume a VL between any pair of VNos in two subsequent layers and the same number of VNos in each layer.



(a) Random (b) Trans-stub

Figure 4.6: An illustration of SNe types.

4.6.3 Cloud infrastructure

In this section, two SNe models are presented.

Random SNe In this model, a random graph with a predefined number of PLs and VLs is generated. The procedure by Broder et al. is used to generate a minimal spanning tree [105], subsequently non-neighboring PMs are interconnected at random, until the desired number of PLs has been reached. An illustration of such a topology is shown in Figure 4.6a.

Trans-stub model SNe Transit-stub network topologies are generated using the Georgia Tech Internetwork Topology Models (GT-ITM) topology generator [106]. An illustration of this SNe model, using 4 transit nodes, is shown in Figure 4.6b. Any two PMs in the transit network are connected by a PL with a probability of 80%. Within a cluster in the stub-network this probability equals 40%. Each PM in the transit network is connected to 2 clusters, each comprising 6 PMs. The total number of PMs in the substrate graph is then equal to thirteen (=transit node + 2×6 stub nodes) times the number of transit nodes. For 1, 2, 3, and 4 transit nodes, the SNe comprises 13, 26, 52 and 104 PMs respectively.

4.6.4 Algorithmic parameters

VAR-GA stops when it has computed 100 generations or if the best solution does not improve during 20 consecutive generations. The algorithms based on subgraph isomorphism detection attempt at most 4 placements per VNo in G^V, as proposed by Lischka et al. The ILP models are solved by Gurobi 6.5.1. All tests were executed on the High-Performance Computing (HPC) core facility CalcUA at the University of Antwerp. Each test used one machine with 20 cores. For the GA, the MongoDB database was hosted on a server at the university, outside the HPC facility. The latency between the HPC and the MongoDB server was 1.740 ms on average.

4.6.5 Key observations

Before analyzing the performance of the algorithms, two metrics must be introduced. First, the placement ratio is the ratio of the number of applications that meet the availability requirement to the total number of application requests. Second, the execution time is the time it takes to execute the placement algorithm. While also an important performance metric, the application response time, has not been simulated. One could incorporate response time requirements by appropriately dimensioning the CPU and bandwidth requirements according to the expected user workload. Additionally, VL latency requirements could easy be added as an additional constraint to the model. Third, the CLF expresses the loading on the cloud environment, i.e. the ratio of total CPU demand of all application requests, to the total amount of available CPU resources:

$$\text{CLF} = \frac{\sum_{s \in S} \sum_{a \in A} I_{a,s} \times \omega_s}{\sum_{n \in N} \Omega_n}. \tag{4.35}$$

In the following, a wide range of parameters is swept. Unless explicitly stated otherwise, for each parameter setting, the CLF is varied from 0.1 to 1, in increments of 0.1. For each CLF level 100 input files are generated, each containing a certain workload and SNe. The simulation results are shown in Figures 4.7 - 4.10, where markers indicate averages and error-bars represent the standard error of the mean.

4.6.5.1 Influence of required availability

Generated workload In this experiment, the workload consists of 10 *random* applications. The VNos of each application are chosen (at random) out of a set of three VNos with a probability of 60%. For each VNo s : ω_s is uniformly distributed in the interval $[0.2; 1]$, and γ_s is uniformly distributed in the interval $[0.75; 1]$. Any two VNos of the same application are interconnected by a VL, requiring a bandwidth which is uniformly distributed in the interval $[0.02; 0.04]$.

Used SNe A random SNe is generated, consisting of 5 PMs and 8 PLs. Each PL has a bandwidth of 1. For each PM n : $\Omega_n \in \{0.5, 2, 10, 50\}$, and $\Gamma_n \in \{1, 1.5, 2\}$. For each PL and PM the failure rate is a uniformly distributed random choice out of the $\{0\%, 2.5\%, 5\%\}$.

Results In Figure 4.7, for a required availability level of 0%, the placement ratio for MOENS-OPT and VAR-GA is very close to optimal (VAR-OPT). The placement ratios for VAR-SUB and MOENS-SUB are about 6% lower. The placement ratios for DRONE-OPT and DRONE-SUB are very low because these algorithms must always find two completely disjoint mappings while there are only 5 PMs in total. As the required availability level increases from 0% to 90%, the placement ratios for the algorithms which are availability-aware (VAR-OPT, VAR-SUB, VAR-GA) remain relatively constant. Additionally, the placement ratios for DRONE-OPT and DRONE-SUB remain constant, as the backup VNe embedding protects against single PM and PL failures. In contrast to that, the placement ratio for MOENS-SUB decreases by 4% and the placement ratio for MOENS-OPT even decreases by 33%. The differences between MOENS-SUB and MOENS-OPT can be attributed to the fact that MOENS-SUB generally uses less PLs

and PMs per application, yielding a higher availability. When the required availability level increases further from 90% to 99% the placement ratios drop significantly for most algorithms. Only for DRONE-OPT and DRONE-SUB the drop is less than 1%, as almost all of the placed applications fulfill the availability requirement. For an availability requirement of 99% there is a huge benefit associated to considering 2 duplicates instead of 1 (for VAR-SUB an increase of 81% in placement ratio), at the cost of an increase of 128% in computation time. Given the drastically improved placement ratio, 2 duplicates will be used in the following experiments, unless explicitly stated otherwise.

While the computation time of DRONE-OPT and MOENS-OPT is below 100 ms, for VAR-OPT it increases dramatically when the required availability goes from 0% to 90%. Most of the configurations with a required availability level of 99% did not finish and were therefore excluded from the graph. While the GA provides a dramatic speedup compared to VAR-OPT for non-zero required availability levels (741x at 90% required availability), the subgraph algorithm is up to 169x faster than the GA. However, the placement ratio of VAR-SUB is up to 14% lower than for VAR-GA. Because finding the exact solution takes too much time for moderate-scale problems, VAR-OPT will be excluded from the remaining experiments.

4.6.5.2 Influence of CPU Load Factor

Generated workload In this setup, 10 *random* applications are generated, comprising 24 VNos in total. Each application is assigned 12 out of those 24 VNos at random. For each VNo $s : \omega_s$ is uniformly distributed in the interval $[0; \omega_{max}]$, and γ_s is uniformly distributed in the interval $[0; \gamma_{max}]$, where

$$\omega_{max} = \frac{CLF_{target}.|N|.2.\overline{\Omega}}{\sum_{a \in A} \sum_{s \in S} I_{a,s}} \tag{4.36}$$

and

$$\gamma_{max} = \Gamma_{max} \times \frac{\omega_{max}}{\Omega_{max}}. \tag{4.37}$$

Equation 4.36 ensures that the expected CLF equals CLF_{target}. Equation 4.37 scales the memory requirements proportionally with the CLF.

Used SNe A trans-stub network comprising 13 PMs, is generated. The PM capabilities are a uniformly distributed random choice of t2, m3, and m4, Amazon EC2 instance specifications [107]. The PL bandwidth is uniformly distributed in $[0; 100]$. Both PM and PL failure rates are uniformly distributed in $[0\%; 1\%]$.

Results The results in Figure 4.8 show that the placement ratio decreases, for all algorithms, as the CLF increases from 0.1 to 1.0. Again, the placement ratio of MOENS-OPT and MOENS-SUB is very low (up to 89% and 71% lower than VAR-SUB) as these algorithms do not introduce any protection. For low CLF values, the performance of DRONE-OPT and DRONE-SUB is comparable to that of VAR-SUB (less than 1% difference). However, when the CLF increases insufficient resources remain to place each VNe twice and the placement ratios of DRONE-OPT and DRONE-SUB are up to 27% and 41% lower than for VAR-SUB. While VAR-SUB performs about 1% better for CLFs up to 0.6, the GA performs 28% better for a CLF of 1.0. This difference

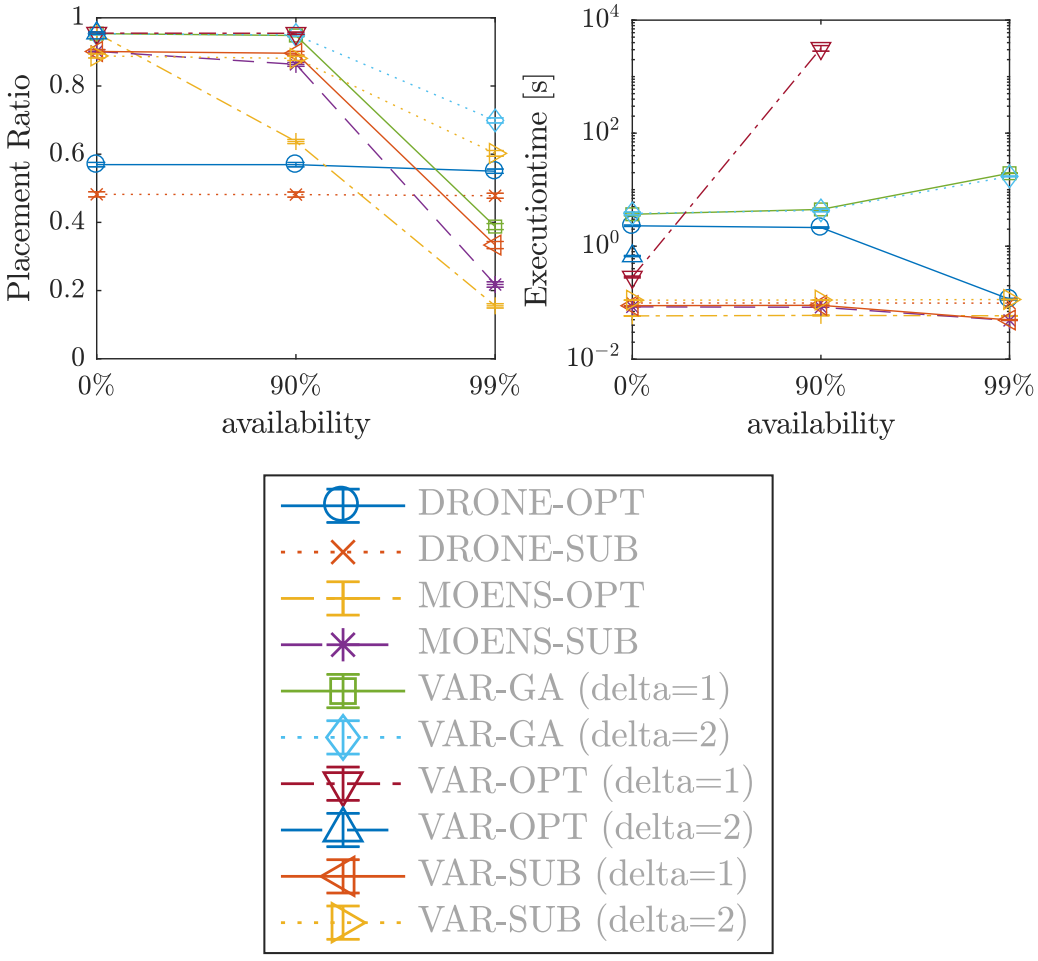


Figure 4.7: Influence of the required availability level.

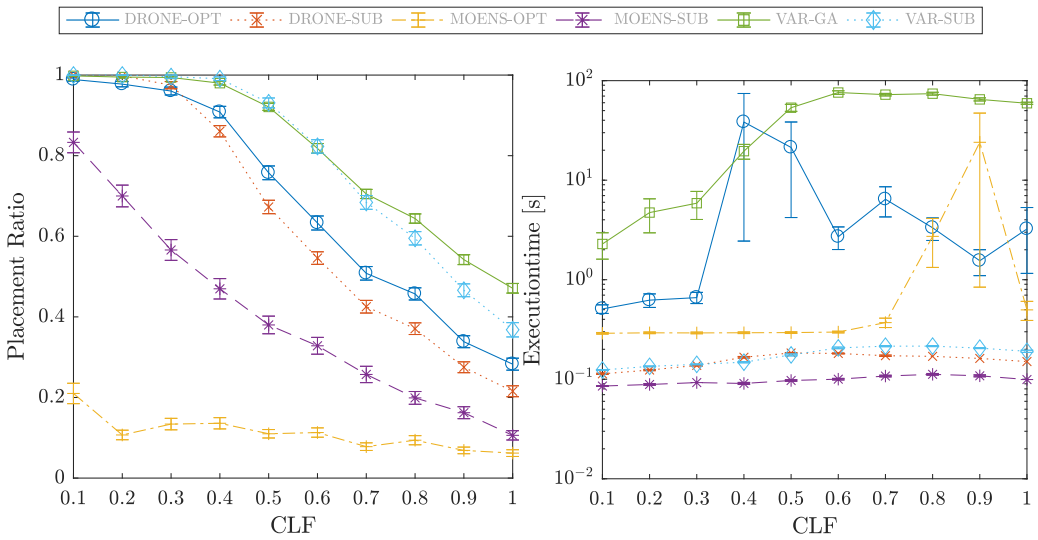


Figure 4.8: Influence of the CLF, for a required availability of 99.0%.

can be explained by how the algorithms handle the application requests. While the subgraph isomorphism algorithm tries to place the applications in a predetermined order and backtracks as soon as an application cannot be placed, the GA tries different orderings of the applications, and when an application cannot be placed, it proceeds to the next one. A similar reasoning can be applied to the differences between DRONE-OPT and DRONE-SUB. For CLF values up to 0.3, DRONE-SUB achieves a higher placement ratio than DRONE-OPT, as typically its embeddings are more compact, realizing a higher availability and ditto placement ratio. However, for CLF values higher than 0.3, DRONE-OPT realizes up to 23% higher placement ratios, as the ILP solver can try different orderings of applications, while DRONE-SUB backtracks as soon as an application cannot be placed.

While the influence of CLF on execution time is not so clear, it is clear that VAR-SUB is much faster than VAR-GA: a speed-up of 400×, up to even 900× is observed. The GA is much slower in our experiments, because of the large overhead in communication with the database. A detailed breakdown of the computation time of the GRECO algorithm shows that up to 93% of the time is spent communicating with the remote DB, even though the workers for one optimization are all running on one and the same PM [C8]. Because of administrative limitations it was not possible to host the DB within the HPC network. Given the dramatic differences in execution time, compared to the other algorithms, only the heuristics based on subgraph isomorphism detection (VAR-SUB, MOENS-SUB, and DRONE-SUB) are included in the remaining experiments.

4.6.5.3 Influence of application requests and SN dimensions

Generated workload For this setup, separate test runs are generated, each considering only *random*, *MapReduce*, or *3 Tier* applications requests. The number of VNos per application is 12, resulting in 5 mappers and 5 reducers, and 4 VNos per layer, in the MapReduce and 3

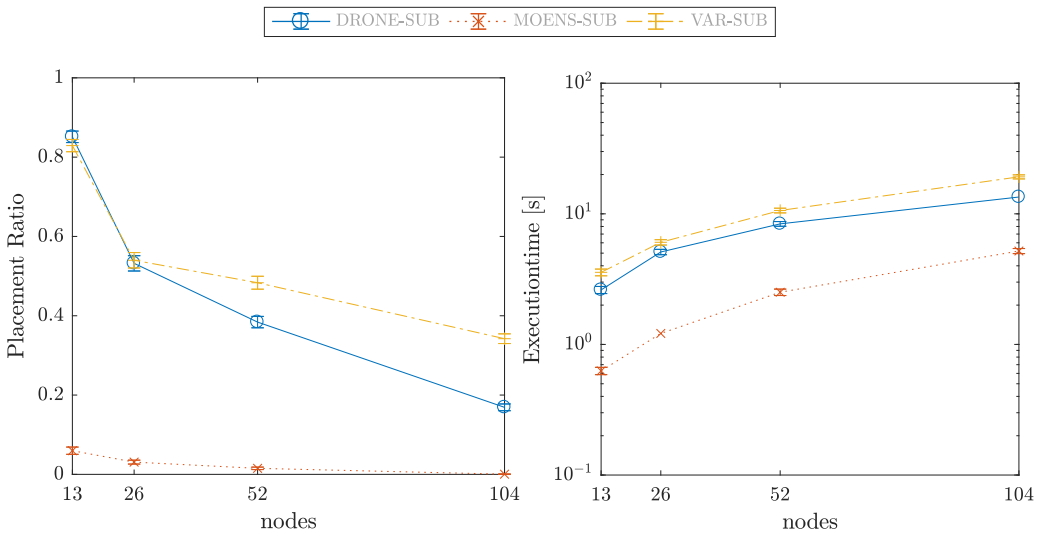


Figure 4.9: Influence of the SNe dimensions for a required availability of 99.9% and application type "random".

Tier model respectively. In Section 4.6.5.2 the number of applications was fixed to 10, and the CLF was increased by generating increasingly CPU heavy applications. However, in this setup the CLF is varied by changing the number of application requests, while keeping the VNo requirements constant. For a CLF_{target} of 0.1, 0.2, and 0.3, the simulation platform generates 10, 20, and 30 applications respectively. Again, for each VNo the CPU and memory requirements are uniformly distributed, with maximum values dictated by Equation 4.36, and 4.37. Also, VL bandwidth requirements are uniformly distributed in $[0; 1]$.

Used SNe A trans-sub model is used with the same specifications as in Section 4.6.5.2, only now the SNe comprises 1, 2, 3, or 4 transit nodes.

Results Figure 4.9 shows that the placement ratio decreases as the SNe size increases. Again, the placement ratio for MOENS-SUB is the lowest of all three algorithms. The placement ratio for 13 and 26 PMs is roughly the same for DRONE-SUB and VAR-SUB. However, when the SNe grows further, then the performance for VAR-SUB is up to 2x better than for DRONE-SUB. An explanation is that as the number of PMs increases, the CPU resources get more fragmented (constant CLF), which makes it harder to always place 2 disjoint duplicates. Additionally, an increased SNe size brings a higher number of reliable PMs. Hence, there is a higher probability that a combination of high available PLs and PMs can be made, avoiding the need for a second duplicate. The execution time for DRONE-SUB is clearly smaller than for VAR-SUB, as the number of possible mappings is smaller (no consolidation possible within application) and as the availability conditions are only checked once (versus before each mapping step).

Figure 4.10 shows that the performance of the algorithm strongly depends on the type of application considered. While the three application types require the same number of

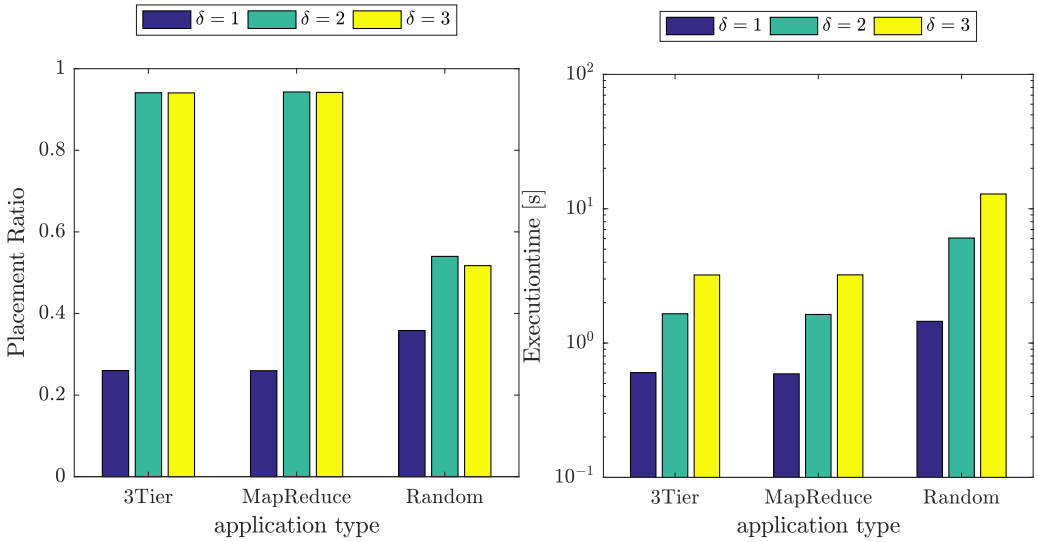


Figure 4.10: Influence of application type for 26 PMs, for a required availability of 99.9%, using VAR-SUB.

VNos, and have the same VNo requirements, the performance for the random and structured applications (MapReduce and 3Tier) differs significantly. The random application type takes significantly longer to place. Additionally, structured applications can benefit more from using more than one duplicate. For the random application type, the placement ratio even decreases slightly when the number of duplicates increases from 2 to 3. Intuitively these observations can be explained by the fact that the VNos of structured applications are easier to consolidate. This effect becomes more pronounced when redundancy levels go up, causing increased competition for CPU resources.

Figure 4.11 shows that the placement ratio decreases and the execution time increases, when the number of application requests goes up. Increasing the number of applications requests causes the memory usage of the subgraph algorithm to increase, as the recursion depth goes up. Therefore, it is good practice to limit the number of mappings to be considered.

Figure 4.12 shows that the placement ratio increases for the structured applications (MapReduce, 3Tier) as the number of PMs increases, while the placement ratio decreases for the random application. This can be explained as follows. On the one hand, when the number of PMs increases, the nodal capabilities get more fragmented, as the CLF is kept constant. On the other hand, as the SNe dimensions increase it become easier to meet the availability requirements, as long as the nodal capabilities are not too fragmented to fit the VNos. It is clear that the computation time increases when the number of PMs increases. Additionally, the computation time for the random application is higher than for the structured applications.

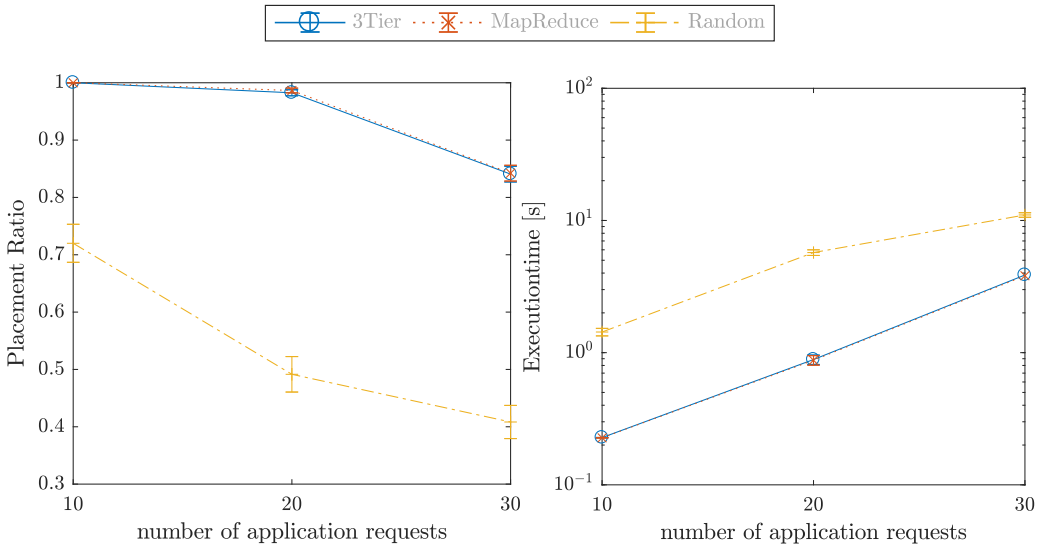


Figure 4.11: Influence of the number of application requests for 26 PMs, a required availability of 99.9%, using VAR-SUB.

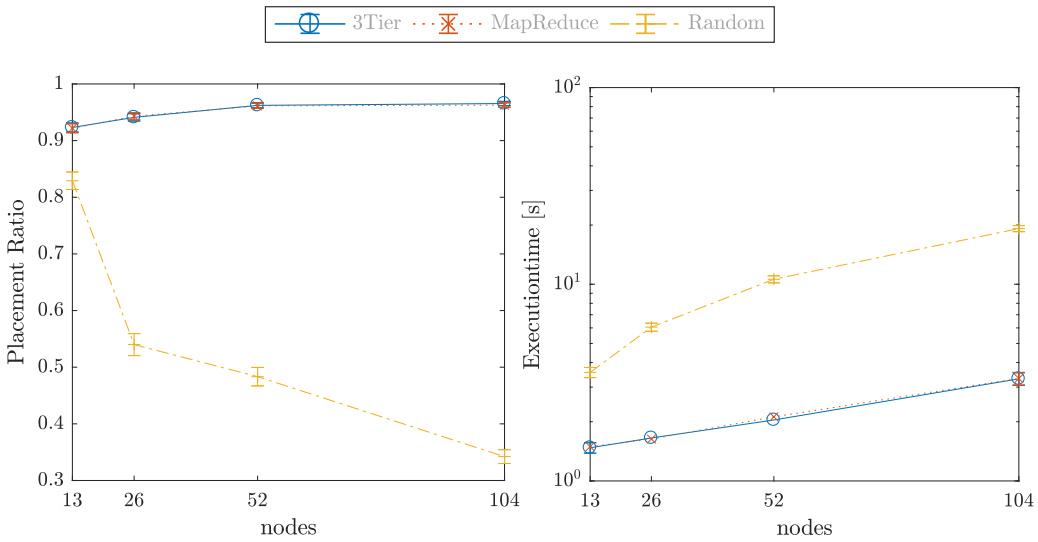


Figure 4.12: Influence of the SNe dimensions for a required availability of 99.9%, using VAR-SUB.

4.7 Results discussion

In the previous section, it was shown that in heterogeneous cloud networks, VAR can bring a dramatic increase in placement ratio, relative to the state of the art, currently lacking an availability model. While exact solution of the ILP formulation scales badly [C5], the GA [C8] can place 10 applications on a SNe comprising 13 PMs within 100 seconds. The newly proposed heuristic based on subgraph isomorphism detection can even place up to 30 applications on a SNe comprising 104 PMs, all within 20 seconds. When the placement algorithm can be offloaded to a (remote) reliable server, then VAR-SUB can be used. If this offloading is not possible, requiring execution of the placement algorithm within the heterogeneous cloud network, then either the fault-tolerant distributed GA must be used, or alternatively the protection methods laid out throughout this work can be applied to VAR-SUB, to make it survive failure of the management nodes.

In reliable cloud environments (or equivalently, under low availability requirements), it is often acceptable to place each VNe only once, and not bother about availability [94]. However, when the frequency of failures is higher (or if availability requirements increase), then one of the following measures should be taken. First, one can improve the availability by placing additional backups, which fail independently of one another. However, this approach works best in homogeneous cloud environments, where one can use the same number of backup VNEs, regardless of the exact PC. In heterogeneous environments, a fixed redundancy level for each application either results in wasted SNe resources, or a reduced placement ratio. The same reasoning can be made for applications, having heterogeneous availability requirements. Second, one can already achieve an increased placement ratio by considering the realized availability during placement (VAR-SUB, $\delta = 1$). In this case, the placement algorithm can place applications with higher availability requirements on more reliable parts of the infrastructure, and at the same time avoid wasting resources on applications whose availability requirements cannot be fulfilled. Third, one can consider the realized availability during placement and decide on the appropriate redundancy level. This decision depends on both the precise availability requirements and the actual PC.

For example, in an Amazon cloud environment, where heterogeneity is limited to multiple generations of servers being used, and resources are virtually infinite, it makes sense to disperse a predefined number of copies across multiple availability zones. However, in an edge-cloud, where computation tasks must be deployed close to the edge of the network, one cannot afford to waste precious resources. While previously intractable, now VAR-SUB can find an intelligent PC, tailored to the specific availability requirements of each application, in a matter of seconds.

When focusing on the placement ratio obtained by VAR-SUB, following observations can be made. First, when the required availability level increases, the placement ratio goes down and the computation time increases. Additionally, more stringent availability requirements require a higher number of duplicates to be used. Second, increased loading of the SNe, either caused by more resource-intensive applications, or an increased number of application requests, decreases the fraction of application requests that can be accepted. While the resource requirements of individual applications have little effect on the execution time, increasing the number of application requests significantly affects computation time. Third, the fraction of application requests that can be accepted largely depends on the type of application requests.

Applications that can be consolidated more easily achieve better placement ratios.

4.8 Conclusion

Cloud environments are becoming increasingly decentralized, leading to a heterogeneous network of micro-clouds which are positioned on the edge of the network and possibly interconnected by best-effort PLs. This heterogeneous environment introduces important challenges for the management of these clouds as the heterogeneity results in an increased failure probability. In this chapter, we study the problem of simultaneous placement of a set of mission-critical applications on such an unreliable network, while guaranteeing a certain level of availability for each application. Three algorithms are presented that make use of intelligent application level replication: an optimal algorithm using an ILP solver (VAR-OPT), and two heuristics. The first heuristic (VAR-GA) is a fault-tolerant distributed GA which uses a distributed pool model to distribute the population. The second (VAR-SUB) is a fast centralized algorithm, based on subgraph isomorphism detection. VAR-GA performs near-optimal for small problem instances and outperforms the subgraph algorithm up to 28% when its placement ratio drops below 0.7. However, when the problem size grows, the VAR-SUB algorithm scales better and becomes the algorithm of choice. While previous solutions were computationally too complex to allow a timely calculation in real-life large-scale environments, the newly presented algorithm based on subgraph isomorphism detection (VAR-SUB), effectively removes this barrier. A detailed performance evaluation shows that, in comparison to algorithms that protect against single PM or PL failure, VAR-SUB can double the placement ratio in cloud environments comprising over 100 PMs, while keeping the time required to calculate the solution under 20 seconds.

Chapter 5

Coordinated allocation of service requests in NFV environments

This work was supported by the University of Antioquia and the 5Guards and FUSE projects. The underlying ideas have been published in [C2].

While Chapters 3 and 4 zoomed in on the QoS-related issues, more specifically, the resilience challenges, connected with the orchestration of NSs in geo-distributed clouds, this chapter focuses on fulfilling the functional requirements of NFV orchestrations in these environments. As described in Chapters 1 and 2, geo-distributed clouds are highly heterogeneous, compared to a centralized cloud whose infrastructure is located within a single datacenter. In geo-distributed cloud environments, NSs typically originate and terminate at specific locations that have limited connectivity. Additionally, bandwidth considerations dictate that VNos that communicate heavily with one another must be placed on PMs that are well-connected. Further in an NFV context, many VNFs have stringent LCs due to their dependence on specialized hardware. This chapter investigates how to compose the NS' VNF-FG and embed it when the infrastructure is highly heterogeneous. More specifically, this chapters focuses on Question III for NSs with a tree topology for their VNF-FG.

5.1 Introduction

NFV is a promising way for service providers to improve configurability of the offered network services. First, NFs and their corresponding flows can be embedded in the SNe in an automated way. Second, the modularity offered by the description of a service as a composition of NFs and their interconnecting VLs, enables tuning of the service's logical structure. A major challenge w.r.t. the management of these environments is the placement of service requests in the physical infrastructure, comprising NFs that can only be instantiated on particular hosts, due to latency, legislation, or hardware-related constraints. These location constraints are known to complicate the resource allocation. This chapter proposes two service placement algorithms with better coordination between the composition and embedding phases.

The remainder of the chapter is structured as follows. Section 5.2 provides an overview of related work. Section 5.3 introduces the combined Service Embedding and Chain Composition (SECC) problem. A formal description of the problem as an ILP is provided in Section 5.4, which can be used to find an exact solution to the problem. The Greedy Chain Selection (GCS) heuristic that can solve larger problem instances is presented in Section 5.5. A detailed performance evaluation is presented in Section 5.6. Finally, Section 5.6.6 concludes the chapter.

5.2 Related work

According to Herrera et al., the main resource allocation challenges related to NFV are VNF chain composition, embedding and scheduling [59]. The VNF-FG provides the logical connectivity (i.e VLs) between virtual appliances (i.e. VNFs) [108]. A VNo is a VNF instance, which can be, e.g., containerized or VM-based. VLs are communication channels required between VNos. VNos typically have processing and memory requirements, while VLs have bandwidth and latency requirements. VNF-FG embedding is very closely related to VNE, which is the task of embedding a VNe request onto a SNe. VNE entails node mapping, which is the mapping of VNos to PMs, and link mapping, which is the routing of VLs through the SNe. The VNE problem on itself has been shown to be NP-hard, and a large body of work has been devoted to finding exact, heuristic and approximate solutions. For a general overview of VNE the reader is referred to [47].

Many emerging network virtualization applications impose LCs, similar to other location-aware applications [78]. Therefore, several researchers have considered those constraints in their VNE approaches. Chowdhury et al. consider that each VNe has to be placed within a certain geographical distance of its preferred physical location [49]. The authors formulate the problem of finding a feasible embedding that maximizes provider revenue as a Mixed Integer Program (MIP). Their proposed heuristics perform coordinated node and link-mapping. Gong et al. consider the problem of finding a minimum-cost VNE subject to additional LCs on VNos [78]. The authors generalize the LCs to the requirement that each VNo can only be instantiated on PMs which are in its candidate set. They reduce the problem to the search for a minimum-cost maximal clique in a compatibility graph.

Recently, researchers have focused on the embedding of SFCs. Ghribi et al. study the problem of finding a minimum-cost embedding of an SFC, subject to multiple resource types and processing and bandwidth constraints [77]. The authors propose a dynamic program that exploits the structuring of the problem in multiple subproblems. Bari et al. reduce the problem of finding a minimum-cost embedding of an SFC, to finding the shortest path from source to sink in an auxiliary graph with associated weights. In each stage of this graph, nodes represent embedding candidates for a particular VNF. The directed edges between stages represent a combination of operational costs. VNFs of the same request cannot be collocated, which is a major limitation, especially for scenarios comprising a large number of light-weight micro-services, combined with sparse computational power, e.g., in a Wireless Sensor Network (WSN) context. Rost et al. consider the problem of embedding multiple services at the same time [67]. They consider both the problem of *admission* and *placement* control, for both SFCs and service cactus graphs. In a service cactus graph two simple cycles share at most

a single node. They propose an approximation algorithm, based on novel ILP formulations together with a decomposition algorithm that enables randomized rounding. In general, the major shortcoming of the aforementioned approaches is that they do not consider service composition.

In contrast, several researchers and industry initiatives did consider the composition problem. For instance, Topology and Orchestration Specification for Cloud Applications (TOSCA) is a data model used by telecom carriers for creating templates or data descriptions of applications and infrastructure for cloud services [109]. It defines the relationships among these services, as well as their operational behavior. Researchers recognized that many telecommunication applications are logically structured as a SFT, i.e. traffic originates at a single root VNF, passes through subsequent intermediate VNF instances that can split the traffic, and then terminates in one or more terminating VNFs [86, 85, 66]. Ocampo et al. studied the problem of generating an SFT with minimum aggregate VL bandwidth demands, subject to precedence constraints in the VNF chaining [66]. The traffic along a VL must pass through a set of predefined VNFs prior to termination. The authors formulate the problem as an MIP, which can be used to synthesize an VNF-FG with minimum aggregate VL bandwidth. The major limitation of the work is that the proposed formulation does not consider the embedding phase. Hence, the practical value of their work is limited, as the VNF-FG with minimum aggregate bandwidth is not necessarily the easiest to embed, or the one that requires the least bandwidth in the SNe. Other works consider both the composition and embedding parts. One can distinguish between uncoordinated and coordinated approaches, based on whether the VNF-FG generation and embedding phases are fully isolated or not. An example of an uncoordinated approach is the work by Mehraghdam et al. [86]. In the composition phase, a greedy heuristic is used to minimize the aggregate VL bandwidth requirements, i.e. VNFs are added to the chain in order of increasing bandwidth consumption. This greedy approach results in an SFT with minimum aggregate bandwidth requirements when all VNFs are required in the chains. The authors assume fixed start and end-points for any service. Additionally, they consider two types of PMs in the SNe, namely switch- and datacenter nodes. The resulting VNF-FG is embedded onto the SNe using an Mixed Integer Quadratically Constrained Program (MIQCP). Beck et al. propose a coordinated approach to the combined composition and embedding problem [85]. The authors propose an approach similar to the VNE heuristic by Lischka et al., based on subgraph isomorphism detection [53]. At each step, their recursive algorithm tries to map all outgoing VLs of the current VNF. The algorithm generates a list of candidate subsequent mappings, each consisting of a possible next VNF and PM. For each VNF the candidate PMs are generated using a BFS algorithm, considering the required bandwidth towards the next VNF, and the maximum path length from the lastly used PM. When an outgoing link can not be mapped then the algorithm backtracks. The algorithm has two major limitations. First, it terminates as soon as a feasible embedding has been found. Hence, when the first chaining that is tried can be embedded, then no other compositions are explored. Second, the algorithm does not take into account the LCs of the VNFs that have to be placed subsequently. Currently, there are no algorithms which exploit flexibility in the service composition, to improve load-balancing. The major contributions of this chapter are

- a formal formulation of the combined SECC problem as an ILP which can be used to find the exact solution; and

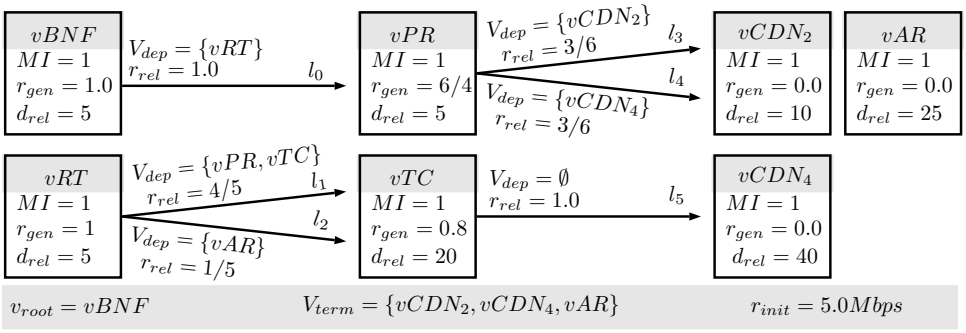


Figure 5.1: Illustration of the SRs.

- a fast heuristic which can be used for larger problem instances.

To the best of our knowledge we are the first to study the impact of LCs on the SECC problem.

5.3 SECC problem formulation

In this section, the SECC problem, which is to find a good PC for a service, given an SNe and SRs, is presented. This problem combines the tasks of VNF-FG composition (Section 5.3.1) and VNF-FG embedding (Section 5.3.2). While the subproblems are introduced separately, the task is to solve both at the same time. Stated otherwise, during composition the constraints and costs related to the embedding should be considered and vice versa. The VNF-FG composition problem is considered as it was introduced by Ocampo et al. [66]. The extension of the problem, to include the VNF-FG embedding, is one of our main contribution. Throughout the remainder of this section, a simplified working example is used to illustrate the key concepts.

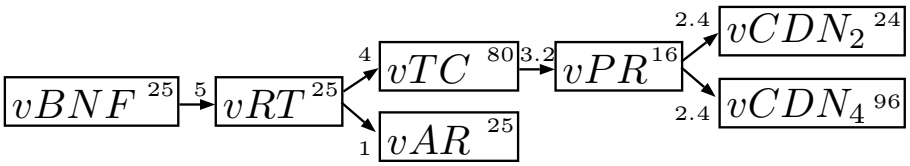
5.3.1 Service composition

The composed VNF-FG must satisfy the SRs. The input parameters to the service composition are listed in Table 5.1. The traffic of the service originates at $v_{root} = vBNF$ (Virtual Basestation Network Function) at rate r_{init}. The processing requirement of VNF $v \in V$ is proportional to its ingress traffic rate (factor $d_{rel}(v)$). The traffic of this VNF must flow through all of its outgoing links $l \in L^v_{out}$. The traffic on this link is proportional to the ingress traffic of the VNF, the rate at which the VNF generates traffic ($r_{gen}(v)$), and the relative traffic rate of l ($r_{rel}(l)$). Hence, the resource consumption depends on the order in which the VNFs are chained. The flow on an outgoing link l must pass exactly once through each of the required VNFs in V^l_{dep}, before termination in one of the terminal VNFs in V_{term}. Ocampo et al. require that the composition comprises at least $M_V(v)$ instances of each VNF $v \in V$.

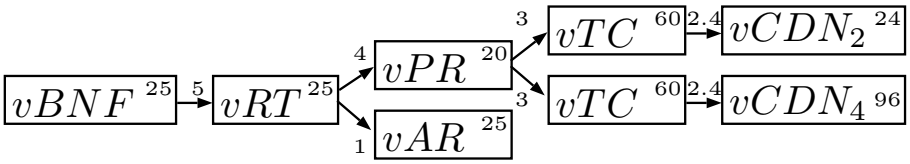
The SRs to the working example are shown in Figure 5.1. This service request comprises 7 VNFs (V) and 6 VLs (L). Five 1 Mbps video streams originate at vBNF, which must be located at Base Station (BS)1. A router (vRT) separates the flow intended for an Augmented Reality (AR) VNF, from the 4 video streams that will be used to create 2 video compositions. These two video

Symbol	Description
V	Set of VNFs in the service.
L	Set of VLs.
$v_{root} \in V$	Root VNF.
$L^v_{out} \subset L$	Set of outgoing VNF links of the VNF $v \in V$
$V^l_{dep} \subset V$	Set of VNFs $\in V$ that the traffic originating from $l \in L$ should pass through before termination.
$V_{term} \subset V$	Set of terminating VNFs.
$r_{init} : \mathbb{R}^+$	Initial data rate arriving at the VNFR.
$r_{gen}(v) : V \rightarrow \mathbb{R}^+$	The ratio of the rate at which VNF v generates traffic, to the ingress data rate at v.
$d_{rel}(v) : V \rightarrow \mathbb{R}^+$	The ratio of the processing requirement of VNF v, to the ingress data rate at v.
$r_{rel}(l) : L \rightarrow \mathbb{R}^+$	The ratio of traffic flowing out l, to the traffic generation rate of VNF $v \in V$.
$M(v) : V \rightarrow \{0, 1\}$	Minimum number of instances for VNF $v \in V$.

Table 5.1: Input parameters to the service composition.



(a) Composition A (C_A): single Transcoder (vTC) instance.



(b) Composition B (C_B): two Transcoder (vTC) instances.

Figure 5.2: Example of two functionally equivalent VNF-FGs.

streams terminate in CDN VNFs $vCDN_2$ and $vCDN_4$, respectively. Each video composition contains 3 out of the 4 video streams, i.e., $\mathscr{F}_1$, $\mathscr{F}_2$, $\mathscr{F}_3$ and $\mathscr{F}_2$, $\mathscr{F}_3$, $\mathscr{F}_4$, respectively. Before termination, the video streams must flow through a proxy (vPR) and transcoding (vTC) VNF. vPR combines two functions. First, it buffers the original stream for both CDN targets, doubling the aggregate bandwidth. Second, it decomposes both resulting streams to filter out one in four substreams for each CDN target, i.e., $\mathscr{F}_4$ and $\mathscr{F}_1$ respectively. Overall, the total bandwidth flowing out of vPR amounts to 150% of the ingress bandwidth to this VNF (= 75% +75%).

In a 2-stage approach to the SECC problem, the outcome of the composition task is a composed service VNF-FG that is ready to be embedded in the next stage. The VNF-FG comprises

Symbol	Description
H	Set of PMs.
E	Set of PLs.
$\phi(v) \subset H$	Set of PMs $\in H$ that can host VNF $v \in V$.
$D(h)$	Remaining processing capability of PM $h \in H$.
$B(h_1, h_2)$	Remaining bandwidth capability of $(h_1, h_2) \in E$.
$T(h_1, h_2)$	Latency introduced by $(h_1, h_2) \in E$.
$t^L(l) : L \rightarrow \mathbb{R}^+$	Upper bound on the delay for $l \in L$.
$t^V(v_{term}) : V_{term} \rightarrow \mathbb{R}^+$	Maximum allowed end-to-end latency between v_{root} and any instance of terminal VNF v_{term}.

Table 5.2: Input parameters related to the service embedding.

VNF instances (V^{inst}), interconnected by directed VL instances (L^{inst}). Figure 5.2 represents two possible VNF-FGs with their corresponding processing and bandwidth requirements, that each satisfy the SRs for the working example. In C_A (Figure 5.2a) the streams are transcoded, prior to caching, while in C_B (Figure 5.2b) the streams are cached, and then transcoded. In C_A, 4 streams are transcoded, while in C_B, 6 streams are transcoded. Hence, C_A requires fewer VNF-FG resources.

5.3.2 Service embedding

The embedding of the VNF-FG must consider the constraints imposed by the SNe. The input parameters to the VNF service embedding are listed in Table 5.2. The SNe comprises PMs (H), interconnected by directed PLs (E). Each PM $h \in H$ has a processing capacity $D(h)$ and each PL $(h_1, h_2) \in E$ has a bandwidth capability $B(h_1, h_2)$ and introduces a delay $T(h_1, h_2)$. A VNF $v \in V$ can only be instantiated on PMs of the required resource type and meeting certain geographical requirements. Both requirements are combined in the constraint that v can only be embedded on $\phi(v) \subset H$. It is assumed that VNFs of the same service request can be collocated on the same machine. It depends on the underlying implementation whether collocated instances of the same VNF run in the same container or VM, or run in separate environments, e.g., for reasons of performance isolation. Typically, memory resources can only be shared among logical VNFs running in the same virtualized environment. Since generally bandwidth and processing requirements are limiting, and memory is (relatively) abundant, we opt to not explicitly consider memory consumption. The inclusion of memory constraints in our algorithms is fairly straightforward. The maximum allowed delay for VL $l \in L$ is given by $t^L(l)$. The Small Cell Forum lists the bandwidth and latency requirements between the BS VNFs [36]. Splits within the physical layer require a latency as low as 0.250 ms, and a bidirectional bandwidth of up to 2.5 Gbps for control and user traffic. At higher layers, the bandwidth and latency requirements are significantly lower. Further, the end-to-end delay from v_{root} to any terminal VNF $v_{term} \in V_{term}$ must not exceed $t^V(v_{term})$.

For the working example, the SNe with indication of the available bandwidth and processing capabilities, is shown in Figure 5.3. A typical 5G infrastructure comprises BS locations, connected to Public Clouds through a MEC network. b_1 through b_4 represent general-purpose

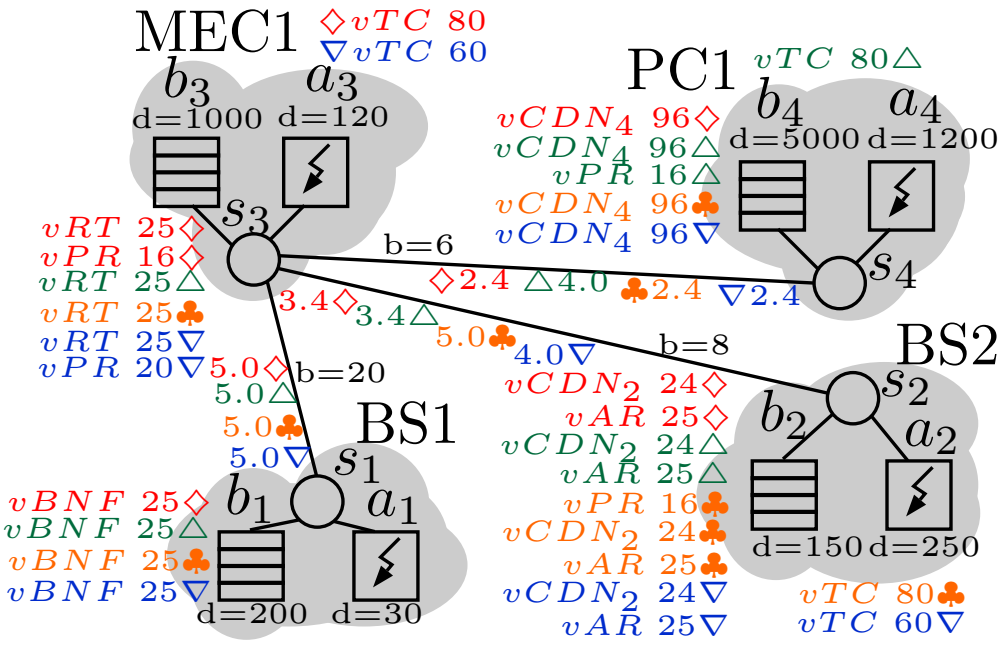


Figure 5.3: Illustration of an SNe in a 5G context, annotated with available bandwidth (b) and processing capabilities (d). The resource consumption of three PCs based on C_A with vTC embedded onto a_3(◇), a_4(△), a_2(♣) and one based on C_B(▽) are shown.

compute power, while a_1 through a_4 represent specialized hardware, that can only execute specific tasks. The LCs for the working example restrict the placement of each source and terminal VNF to a single PM. Hence, $\phi(vBNF) = \{b_1\}$, $\phi(vAR) = \{b_2\}$, $\phi(CDN_2) = \{b_2\}$ and $\phi(CDN_4) = \{b_4\}$. Further, vTC can only be executed on nodes with accelerator capabilities, $\phi(vTC) = \{a_1, a_2, a_3, a_4\}$. Both vRT and vPR can be executed on any of the general-purpose nodes, $\phi(vRT) = \phi(vPR) = \{b_1, b_2, b_3, b_4\}$.

The result of the SECC problem is a PC, which is a composed VNF-FG combined with a valid VNE. This VNE entails a mapping of each VNF instance to a PM and a mapping of each VL instance to a path in the SNe. Consider the placement of two requests of the working example service type, i.e., s_0 and s_1. Four possible PCs, i.e., ♢, ♣, △ and ▽, are shown in Figure 5.3. ♢, ♣ and △ are based on C_A, while ▽ is based on C_B. It is impossible to place both services at the same time using C_A. First, {♢, ♢} violates the processing capability of a_3 as $80+80 > 120$. Second, {♢, △} violates the bandwidth capacity of (s_3, s_4) as $2.4+4 > 6$. Third, both {♢, ♣} and {△, ♣} violate the bandwidth capacity of PL (s_3, s_2) as $3.4+5 > 8$. In contrast, two PCs based on C_B can be placed together, i.e. {▽, ▽}. Furthermore, a combination of C_A and C_B can also be placed, i.e., {♢, ▽} or { ♣, ▽}. This small example illustrates that both composition and embedding procedures impact the resource consumption and acceptance ratio in the SNe.

5.4 Exact algorithm (OPT-SECC)

This section describes OPT-SECC, an optimal algorithm to solve SECC. In our approach, first all valid VNF chains, together with their corresponding VL bandwidths and VNF processing requirements are generated. Subsequently, these chains are used as input to an ILP, which can find the minimum average-load embedding, satisfying the requirements of Sections 5.3.1 and 5.3.2. The resulting PC is guaranteed to be optimal in the objective function, for two reasons. First, any valid VNF-FG can be described as a combination of the generated chains. Second, the ILP simultaneously selects a valid subset of these chains and embeds them.

5.4.1 Chain generation

Recall that for a PC to be valid, exactly one of the possible VNF-FG configurations, satisfying the SRs must be selected and embedded, i.e., either Figure 5.2a or 5.2b for the working example. For the embedding of this VNF-FG, each comprising VNF instance must be mapped to a PM and all its egress VL instances must be routed through the SNe. Since each service request has a single root VNF instance and all other instances have a single ingress VL, the topology of any valid VNF-FG is a tree. The exact algorithm proposed in this section is based on the reformulation of the combined SECC problem on an augmented tree, instead of directly on the input parameters in Table 5.1. The augmented tree represents all possible VNF-FGs satisfying the SRs, without needing to explicitly enumerate them. By construction, any valid VNF-FG will be a subgraph of this augmented tree. The procedure to construct this augmented tree is presented at the end of this subsection.

Strictly speaking, each vertex $\theta \in \Theta$ in the augmented tree is a VNF instance and its ingress link is a VL instance. Only a subset of the VNF instances represented in the augmented tree

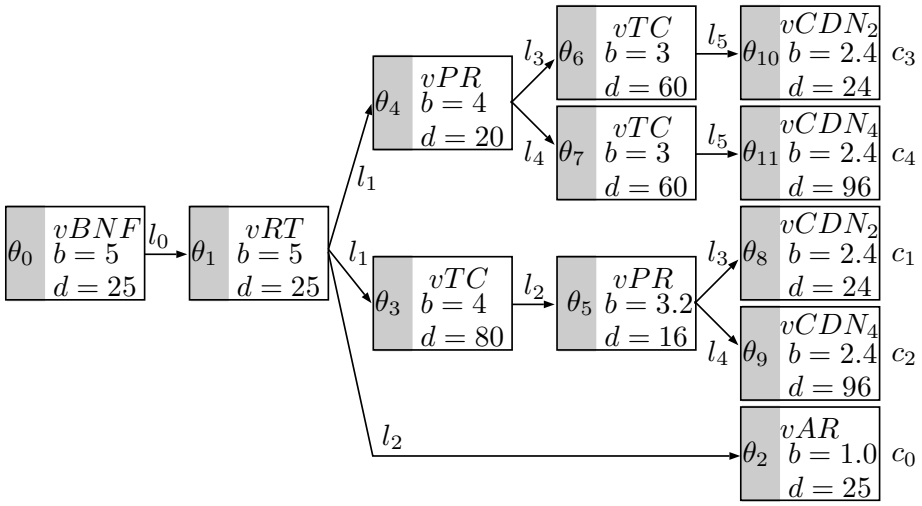


Figure 5.4: Augmented tree resulting from application of Algorithm 5.1 to the service request depicted in Figure 5.1. The valid compositions are $C_A = \{c_0, c_1, c_2\}$ and $C_B = \{c_0, c_3, c_4\}$.

will eventually end up in the VNF-FG and be instantiated. We will refer to the vertices in the augmented tree as augmented nodes, to emphasize this distinction.

Given this representation, the SECC problem can be reformulated in terms of selecting and embedding augmented nodes. The leaf augmented nodes correspond to terminating VNF instances. The sequence of augmented nodes and augmented links from the augmented root VNF to a leaf augmented node will be referred to as a (service function) chain. Each chain in the augmented tree satisfies the precedence constraints of its composing VL, i.e. for each VL l in the chain, its required VNFs appear in the chain, exactly once, and not before l. For the working example, the augmented tree is shown in Figure 5.4.

Each augmented VNF $\theta \in \Theta$ has a corresponding VNF $v(\theta)$ and corresponding ingress VL $l(\theta)$. For instance, $v(\theta_1) = vRT$. The processing and bandwidth requirement of $v(\theta)$ and $l(\theta)$ are given by $d(\theta)$ and $b(\theta)$ respectively. If $\theta_2 \in \Theta$ is a child of $\theta_1 \in \Theta$, then this relation implies that the chaining constraints allow, $v(\theta_2)$ to be chained after $v(\theta_1)$, via $l(\theta_2)$. Since each VNF-FG satisfying the chaining requirements is a subgraph of the augmented tree, each possible VNF-FG is formed by a subset of the chains C in the augmented tree. For instance, C_A is formed by the union of c_0, c_1 and c_2, which comprises θ_0, θ_1, θ_2, θ_3, θ_5, θ_8, θ_9 and their ingress augmented VL instances.

The remaining composition requirements for the VNF-FG can be formulated in terms of the augmented tree. If an augmented node is selected, then for each of its egress VLs exactly one child needs to be selected as target of this VL. For instance, if vRT is part of the composition, then its egress link l_3 needs to be connected to an instance of either vPR (θ_4) or vTC (θ_3), but not both. vRT (θ_4) can only be connected via l_1 to an instance of vAR (θ_2). Compared to the approach by Ocampo et al., the generated chains satisfy the precedence requirements by construction, greatly simplifying the ILP formulation, as the number of decision variables and constraints is drastically reduced [66].

Algorithm 5.1 Depth First Search (DFS) chain construction algorithm.

1: **var** $C, V_{term}, v_{root}, V, L, \theta_{cur}, r_{init}$
2: $C = [\,]$
3: $V_{req} = \{\}$
4: θ_{cur} =new $\theta(\varnothing, v_{root})$
5: $b(\theta_{cur}) \leftarrow r_{init}$
6: $d(\theta_{cur}) \leftarrow r_{init} \times d_{rel}(v_{root}))$
7: **procedure** SEARCH(V_{req})
8: $\Theta_{can} \leftarrow$ GENC(V_{req})
9: $order(\Theta_{can})$
10: **for each** $\theta_{can} \in \Theta_{can}$ **do**
11: $children(\theta_{cur}) \leftarrow \{children(\theta_{cur}), \theta_{can}\}$
12: **if** $v(\theta_{can}) \in V_{term}$ **then**
13: $C \leftarrow \{C, ancestors(\theta_{can})\}$
14: **else**
15: $V_{req}^{new} \leftarrow \{V_{req}, V_{dep}^{l}\} \setminus v(\theta_{can})$
16: $\theta_{cur} \leftarrow \theta_{can}$
17: SEARCH(V_{req}^{new})
18: $\theta_{cur} \leftarrow parent(\theta_{can})$
19: **end if**
20: **end for**
21: **if** $\exists l \in L_{out}^{v(\theta_{cur})} | \nexists \theta \in children(\theta_{cur}) : l(\theta) == l$ **then**
22: $children(parent(\theta_{cur})) \leftarrow children(parent(\theta_{cur})) \setminus \theta_{cur}$
23: $C \leftarrow C \setminus \{c \in C | \theta_{cur} \in \Theta(c)\}$
24: **end if**
25: **return**
26: **end procedure**

The DFS procedure to construct all valid chains is described in Algorithm 5.1. The algorithm keeps track of the valid chains C. At each recursion, the state of the algorithm is determined by following variables. The current augmented VNF, θ_{cur} represents the augmented VNF that has been added to the current chain most recently. The VNFs that must still be added to the current chain are stored in V_{req}. Each chain originates at the root VNF, which has no ingress VL (Line 4). The algorithm considers the subset of VNF and VL combinations that can be chained after θ_{cur}. The order in which the augmented VNFs are considered depends on the $order$ function (Line 9). Here, the augmented VNFs are ordered according to increasing bandwidth requirements. Each augmented VNF candidate θ_{can} is added as a child to its parent θ_{cur}. If θ_{can} corresponds to a terminating VNF, then the chain from the augmented root VNF to θ_{can} is added to C (Line 13). Here, $ancestors(\theta_{can})$ returns the subchain of augmented VNFs from the augmented root VNF to, and including, θ_{can}. If θ_{can} does not correspond to a terminating VNF, then a new set of required VNFs V_{req}^{new} is generated and the algorithm recurses (Line 17). V_{req}^{new}, includes the previously required VNFs in V_{req}, and the additional ones required by $l(\theta_{can})$, which are not equal to $v(\theta_{can})$ (Line 15). When all augmented VNF candidates have been explored, then, before backtracking, the algorithm verifies if a valid chain, containing θ_{cur}, exists for each of the required outgoing VLs (Line 21). If not all outgoing VNF links can be chained to θ_{cur}, then θ_{cur} is removed as a child of its parent (Line 22) and all chains in C containing θ_{cur} are removed (Line 23). By construction, for any augmented non-root VNF in the tree, a valid chaining exists for all of its corresponding outgoing VLs. Consequently, a feasible composition is guaranteed to exist when the algorithm terminates and the augmented root VNF has children.

Algorithm 5.2 Augmented VNF candidate generation.

1: **var** $v_{root}, V_{term}, V, L, \theta_{cur}$
2: **procedure** GENC(V_{req})
3: $\Theta_{can} \leftarrow \emptyset$
4: **for each** $l \in L_{out}^{v(\theta_{cur})}$ **do**
5: $V'_{req} \leftarrow \{V_{req}, V_{dep}^{l}\}$
6: **if** $V'_{req} \subset V_{term}$ **then**
7: $V_{can} \leftarrow V'_{req}$ ▷ Chain terminates
8: **else**
9: $V_{can} \leftarrow \{v \in V'_{req} \setminus V_{term} : \nexists \theta \in ancestors(\theta_{cur}) | v(\theta) = v\}$ ▷ VNFs pending
10: **end if**
11: **for each** $v \in V_{can}$ **do**
12: $\theta_{new} \leftarrow$new $\theta(l, v)$
13: $b(\theta_{new}) \leftarrow b(\theta_{cur}).r_{gen}(v(\theta_{cur})).r_{rel}(l)$
14: $d(\theta_{new}) \leftarrow b(\theta_{new}).d_{rel}(v)$
15: $parent(\theta_{new}) \leftarrow \theta_{cur}$
16: $\Theta_{can} \leftarrow \{\Theta_{can}, \theta_{new}\}$
17: **end for**
18: **end for**
19: **return** Θ_{can}
20: **end procedure**

Not all VNFs in V can be added to the current subchain, as a chain can contain at most one instance of each VNF and the chain cannot terminate before all VNF dependencies have been satisfied.

Algorithm 5.2 generates the set of augmented VNF candidates, Θ_{can}, that can be chained after θ_{cur}. For each outgoing link l of θ_{cur} (Line 4), the set of VNFs that the traffic on l must go through to fulfill the combined VNF dependencies is stored in V'_{req} (Line 5). When only terminating VNFs are required, then any of the required non-visited VNFs can be added to the chain (Line 7). In case non-terminal VNFs are required, then only these can be added after θ_{cur} (Line 9). Next, for each valid augmented VNF candidate, the bandwidth (Line 13) and processing (Line 14) requirements are calculated based on the requirements of its parent θ_{cur}. Finally, its parent relation is set (Line 15) and the candidate is added to candidate set Θ_{can} (Line 16).

Symbol	Description
V	Set of VNFs for the service.
C	Set of chains that meet the dependence constraints.
$v_{term}(c) : C \rightarrow V_{term}$	Terminal VNF v_{term} corresponding to chain $c \in C$.
Θ	Set of augmented VNF nodes in the pruned chain search tree.
$v(\theta) : \Theta \rightarrow V$	VNF corresponding to $\theta \in \Theta$.
$l(\theta) : \Theta \rightarrow L$	Ingress VNF link to $\theta \in \Theta$.
$\Theta(c) : C \rightarrow \Theta$	Set of augmented VNFs in chain c.
$d(\theta) : \Theta \rightarrow \mathbb{R}^+$	The processing requirement of augmented VNF $\theta \in \Theta$.
$parent(\theta) : \Theta \rightarrow \Theta$	Parent augmented VNF of $\theta \in \Theta$.
$children(\theta) : \Theta \rightarrow \Theta$	Children of $\theta \in \Theta$.
$b(\theta) : \Theta \rightarrow \mathbb{R}^+$	The ingress bandwidth of $\theta \in \Theta$, i.e. $b(l(\theta))$.

Table 5.3: Input parameters to the service composition for the ILP.

Symbol	Description
u_c	Binary variable indicating if chain $c \in C$ is part of the composition.
X_θ	Binary variable indicating if augmented VNF $\theta \in \Theta$ is used.
$x_{\theta,h}$	Binary variable indicating if augmented VNF $\theta \in \Theta$ is embedded onto PM $h \in H$.
$y_{\theta,(h_1,h_2)}$	Binary variable indicating if the augmented VL instance to augmented VNF θ, flows over $(h_1, h_2) \in E$.
$\Pi_{\tilde{\theta},h}$	Binary variable linearizing the product $x_{parent(\tilde{\theta}),h} X_{\tilde{\theta}}$.

Table 5.4: Decision variables of the ILP.

5.4.2 Integer Linear Program

In this section, we provide a formal description of our problem as an ILP [110]. Table 5.3 lists the input parameters related to the VNF chains and service composition. These parameters result from the chain generation algorithm described in Section 5.4.1. The input parameters related to the service embedding are the same as for the original problem (Table 5.2). The decision variables are listed in Table 5.4.

The remainder of this section is structured as follows. First, the constraints related to the VNF-FG composition and VNE are provided. Then, the objective function to the optimization problem is given.

5.4.3 Composition constraints

Each VNF $v \in V$ must be instantiated at least $M(v)$ times, placing a lower bound on the number of chains containing v in the VNF-FG.

$$M(v) \leq \sum_{c \in C : \exists \theta \in \Theta(c) | v(\theta) = v} u_c : \forall v \in V, \tag{5.1}$$

Decision variable u_c indicates if $c \in C$, is selected as part of the composition. If an augmented VNF is used, then each of its outgoing VLs is embedded exactly once. Conversely, if an augmented VNF is used, then so is its parent.

$\forall \theta \in \Theta, l \in L_{out}^{v(\theta)}$:

$$X_\theta = \sum_{\tilde{\theta} \in children(\theta) : l(\tilde{\theta}) = l} X_{\tilde{\theta}} \tag{5.2}$$

If chain $c \in C$ is selected, then all the augmented VNFs that are part of c must be used.

$$u_c \leq X_\theta : \forall c \in C, \theta \in \Theta(c) \tag{5.3}$$

5.4.4 Embedding constraints

If an augmented VNF θ is used, then it is embedded onto exactly one PM $h \in H$.

$$X_\theta = \sum_{h \in H} x_{\theta,h} : \forall \theta \in \Theta \tag{5.4}$$

The processing requirements for the VNFs are assumed additive. If an augmented VNF θ is embedded onto $h \in H$, then its processing requirements $d(\theta)$ must be allocated. For each PM $h \in H$, the total processing requirements cannot exceed the node's remaining capabilities.

$$\sum_{\theta \in \Theta} d(\theta) x_{\theta,h} \leq D(h) : \forall h \in H \tag{5.5}$$

VNF $v \in V$ cannot be instantiated on PMs that are not in its candidate set $\phi(v) \subset H$.

$$x_{\theta,h} = 0 : \forall \theta \in \Theta, h \in H \setminus \phi(v(\theta)) \tag{5.6}$$

If augmented VNFs are used, then their interconnecting VLs must be routed through the SNe. MCF constraints imply that for any VL the net number of flows leaving a certain PM h_1 depends on the embedding location of the source and target VNF of this VL. This number equals *1* if solely the VL source is hosted on h_1, *-1* if both are located on h_1 or *0* when either both or none are placed on h_1.

$\forall h_1 \in H, \theta \in \Theta, l \in L_{out}^{v(\theta)}, \tilde{\theta} \in \{\hat{\theta} \in children(\theta) : l(\hat{\theta}) = l\}$:

$$\sum_{(h_1,h_2) \in E} y_{\tilde{\theta},(h_1,h_2)} - \sum_{(h_2,h_1) \in E} y_{\tilde{\theta},(h_2,h_1)} = \Pi_{h_1,\tilde{\theta}} - x_{\tilde{\theta},h_1}, \tag{5.7}$$

where $y_{\theta,(h_1,h_2)}$ is a binary variable indicating if the ingress traffic of augmented VNF θ is routed along (h_1, h_2). Note that a binary value of $y_{\theta,(h_1,h_2)}$ results in the mapping of a VL

to at most a single path in the SNe. When the SNe supports path splitting, then $y_{\theta,(h_1,h_2)}$ can assume any value in $[0;1]$.

$\Pi_{\tilde{\theta},h}$ is a binary variable indicating if $parent(\tilde{\theta})$ is hosted on h and augmented VNF $\tilde{\theta}$ is used. Hence, $\Pi_{\tilde{\theta},h} = x_{parent(\tilde{\theta}),h} X_{\tilde{\theta}}$, which is linearized by $\forall \tilde{\theta} \in \Theta | v(\tilde{\theta}) \neq v_{root}, h \in H$:

$$\Pi_{\tilde{\theta},h} \leq x_{parent(\tilde{\theta}),h}, \tag{5.8}$$

$$\Pi_{\tilde{\theta},h} \leq X_{\tilde{\theta}}, \text{and} \tag{5.9}$$

$$\Pi_{\tilde{\theta},h} \geq x_{parent(\tilde{\theta}),h} + X_{\tilde{\theta}} - 1 \tag{5.10}$$

If for VNF $\theta \in \Theta$, the ingress VL is routed along PL $(h_1, h_2) \in E$, then its required bandwidth $b(\theta)$ must be allocated. The total bandwidth consumption on a PL cannot exceed the total available bandwidth $B(h_1, h_2)$.

$$\sum_{\theta \in \Theta} b(\theta) y_{\theta,(h_1,h_2)} \leq B(h_1,h_2) : \forall (h_1,h_2) \in E \tag{5.11}$$

The delay constraints are twofold. First, the delay of VL $l \in L$ cannot exceed $t^L(l(\theta))$. For each augmented link instance, the delay of its ingress VL is the sum of the delays of the PLs that it is routed over. $\forall (h_1,h_2) \in E, \theta \in \Theta | v(\theta) \neq v_{root}$:

$$\sum_{(h_1,h_2) \in E} y_{\theta,(h_1,h_2)} T(h_1,h_2) \leq t^L(l(\theta)) \tag{5.12}$$

Second, the end-to-end delay between v_{root} and any instance of $v_{term} \in V_{term}$ cannot exceed $t^V(v_{term})$. $\forall v_{term} \in V_{term}, c \in C | v_{term}(c) = v_{term}$:

$$\sum_{\Theta(c)} \sum_{(h_1,h_2) \in E} y_{\theta,(h_1,h_2)} T(h_1,h_2) \leq t^V(v_{term}) \tag{5.13}$$

5.4.5 Objective function

The objective of the resource allocation is to minimize resources consumption, weighted by the scarcity of said resource.

$$\mathbb{L} = \sum_{\theta \in \Theta} \sum_{h \in H} \frac{d(\theta) x_{\theta,h}}{D(h)} + \sum_{\theta \in \Theta} \sum_{(h_1,h_2) \in E} \frac{b(\theta) y_{\theta,(h_1,h_2)}}{B(h_1,h_2)} \tag{5.14}$$

The objective is to minimize $\mathbb{L}$ subject to Equations 5.1 - 5.13.

5.4.6 Discussion

Relative to the formulation by Ocampo et al. [66], our augmented tree approach greatly simplifies the ILP formulation for the chaining and composition for several reasons. First, their formulation considers all acyclic VNF chainings and verifies precedence constraints using the ILP. Second, in our approach, only valid chainings are considered, which greatly reduces the number of chains to be considered. Third, precomputation of the valid chainings reduces the

constraints related to the composition problem to Equations 5.1, 5.2 and 5.3, compared to their 25 equations.

An upper bound on the number of augmented VNFs is

$$|\Theta| \leq |V|(L_{out}^{max}.V_{can}^{max})^{|V|-1}, \tag{5.15}$$

where $L_{out}^{max} = \max_{v \in V} |L_{out}^{v}|$ is the maximum number of outgoing VL for any VNF in V, and V_{can}^{max} is the maximum number of candidates considered for chaining after any VNF. Hence, for applications with a lot of flexibility in the chaining, the number of augmented VNFs quickly becomes unmanageable and the optimal solution can no longer be determined within reasonable computation time.

However, to limit the size of the tree, one can limit the number of children for each augmented VNF corresponding to each outgoing link, while pruning the tree in Algorithm 5.1. This limitation is considered out of scope.

The minimum-bandwidth VNF-FG composition problem can be reduced to finding a minimum-cost maximal clique in a graph $G(\Theta, \Psi)$, where nodes Θ are augmented VNFs and an edge $\psi \in \Psi$ indicates if its endpoints are compatible. Any two augmented VNFs are incompatible i.f.f., they connect to the same parent via the same VL $l \in L$, or if their parents are incompatible. While a minimal clique can be generated in linear time, finding a minimum-cost maximal clique is NP-hard [111]. The combined SECC problem is even harder than the minimum-bandwidth VNF-FG composition problem, because the costs now depend on the embedding.

5.5 Greedy Chain Selection Heuristic

In this section, a GCS algorithm that iteratively adds chains and embeds them, is proposed for the SECC problem. This procedure is based on a greedy heuristic for set cover, that at each iteration selects a set that has the highest utility [112]. The main difference compared to this greedy set cover heuristic, is that the universe of elements that need to be covered (U) grows as more chains are selected. Additionally, since several augmented nodes are not compatible, a list of blacklisted augmented VNFs (Θ_{black}), that cannot be added to the composition, must be maintained. This section is structured as follows. First, the main routine of the greedy algorithm is presented. Then, the subroutine to embed the SFCs is introduced. Finally, an illustration of the algorithm concludes this section.

5.5.1 Greedy Chain Selection

The GCS algorithm is described in Algorithm 5.3. The algorithm requires the following input: the set of feasible chains (C), the set of PMs (H) and PLs (E), v_{root}, the available bandwidth (B) and processing (D) resources and the set of all-pair k-shortest PPs ρ in graph $G(H, E)$. The k-ShPs between any two nodes are generated using the algorithm proposed by Martins et al. [113]. The algorithm can be configured to either compute all paths at once, or rather compute them incrementally. $\rho_{h_1,h_2,\varrho}$ is the resulting ϱ^{th} ShP from h_1 to $h_2 : h_1 \in H, h_2 \in H, \varrho \in \{0, 1, \ldots, k-1\}$. The selected chains are stored in $C_{res} \subset C$, which is

Algorithm 5.3 GCS-SECC algorithm.

```
1:  var C_res, H, E, B, D, ρ
2:  procedure GCSEMBEDDING(C, v_root)
3:      C_res ← ∅
4:      C_can ← ∅
5:      M_res ← ∅
6:      Θ_dirty ← Θ(C)
7:      Θ_black ← ∅
8:      U ← {(X_θ, l_out) : θ ∈ Θ(C), v(θ) = v_root, l_out ∈ L_out^{v_root}}
9:      while U ⊊ cover(C_res, U) do
10:         for each c ∈ C_can do
11:             if c ∩ Θ_dirty ≠ ∅ then
12:                 G_c ← AG(c, M_res)
13:                 M_c ← ME(G_c, 0)
14:             end if
15:         end for
16:         c_best ← arg max_{c∈C\C_can} { |cover(c,U)\cover(C_res,U)| / cost(M_c) }
17:         κ ← 1
18:         while !valid(M_{c_best}) do
19:             M_{c_best} ← ME(G_{c_best}, κ)
20:             κ ← κ + 1
21:             if κ == K then
22:                 M_res ← ∅
23:                 return false
24:             end if
25:         end while
26:         Θ_black ← {θ_1 ∈ Θ(C) : ∃θ_2 ∈ c_best | parent(θ_1) = parent(θ_2), l(θ_1) = l(θ_2)}
27:         M_res ← M_res ∪ M_{c_best}
28:         C_res ← {C_res, c_best}
29:         C_can ← C_can \ {c ∈ C_can : Θ(c) ∩ Θ_black ≠ ∅}
30:         U ← U ∪ {(θ, l_out) : θ ∈ Θ(c_best), l_out ∈ L_out^{v(θ)}}
31:     end while
32:     update B, D
33:     return true
34: end procedure
```

initialized on Line 3. Afterwards, the node and link mappings M_{res} are initialized (Line 5). Subsequently, the set of elements that require covering, U, is initialized (Line 8). The universe initially contains all combinations of the root augmented VNF with its outgoing VLs. Additional composition requirements, as per Equation 5.1, can be included here too.

While the selected chains do not cover all elements in U (Line 9), the algorithm iteratively selects the next chain with the highest utility and tries to embed it. In each iteration of this loop, the chain in C_{can} with the highest utility, c_{best} is selected. The utility is defined as the ratio of the number of newly covered elements in U, to the (estimated) embedding cost of this chain. The cost of a chain must be updated if any of its augmented VNFs is labeled *dirty*, i.e. its mapping was altered in the previous iteration (Line 11). To estimate the cost of a chain, an Auxiliary Graph $G_c = AG(c, M_{res})$ is generated, considering the existing mappings in M_{res} (Line 12). Then, G_c is used to generate a minimum-cost embedding M_c (Line 13). $ME(G_c, \kappa)$ is the procedure to calculate the κ^{th} minimum-cost embedding of chain c, using AG G_c. Both procedures are explained in detail, in Section 5.5.2.

The cost of a mapping M_c is determined by Equation 5.14. If $M(c_{best})$ violates one or more resource constraints, or the end-to-end delay constraint (Line 18), then the next minimum-

cost embedding is generated (Line 19). If no feasible embedding can be generated within K attempts, then the mapping is undone (Line 23) and the request is declined (Line 22). If this SFC can be embedded, then the set of augmented VNFs violating Equation 5.2 is updated (Line 26). Furthermore, the mapping M_{res} is updated with the new mappings in $M_{c_{best}}$. The set of selected chains (C_{res}) is updated with c_{best}. Then, the chains containing blacklisted augmented VNFs are removed from the set of candidate chains for the next iteration (C_{can}) (Line 29). Lastly, all outgoing VLs of the augmented VNFs in c_{best} need to be covered as per Equation 5.2 (Line 30).

Ultimately, if all required elements are present in C_{res} (Line 9), then the algorithm terminates successfully (Line 33).

5.5.2 Minimum-cost SFC embedding

The K least-cost embeddings, for an SFC $c \in C$, are calculated in two steps. First, an AG $G_c \leftarrow AG(c, M_{res})$ is generated. Figure 5.5 shows the AG, corresponding to the embedding of c_4. This chain is part of the augmented VNF tree shown in Figure 5.4, corresponding to the working example. For an augmented VNF $\theta \in \Theta(c)$ that is not (yet) in M_{res}, its corresponding stage contains all candidate PMs in $\phi(v(\theta))$. Stages corresponding to augmented VNFs in M_{res} contain only the mapping defined in M_{res}. Therefore, the embedding possibilities for a particular service decrease as more chains are selected. Additionally, nodes S and T are introduced, representing the virtual source and sink of the chain, respectively. The cost of mapping VL (θ_1, θ_2) to PP $\rho_{h_1,h_2,\varrho} \in \rho$ is $\forall \theta_2 \in \Theta, \theta_1 = parent(\theta_2)$:

$$\begin{aligned} \mathscr{C}(\theta_2, \rho_{h_1,h_2,\varrho}) = & \frac{d(\theta_1)}{D(h_1)} . \mathbf{1}_{\Theta(C \setminus C_{res})}(\theta_1) \\ & + \sum_{e \in \rho_{h_1,h_2,\varrho}} \frac{b(\theta_2)}{B(h_1,h_2)} . \mathbf{1}_{\Theta(C \setminus C_{res})}(\theta_2), \end{aligned} \tag{5.16}$$

where $\mathbf{1}_A(x) : X \rightarrow \{0,1\}$, with $A \subset X$, is an indicator function which equals 1 i.f.f. $x \in A$. Equation 5.16 only explicitly considers the processing cost of the VL's source VNF (θ_1) onto h_1. The processing cost of hosting the VL's target VNF (θ_2) onto h_2 is attributed to the subsequent VL mapping with source VNF θ_2. The choice to assign the processing cost of a VNF in an SFC fully to either its egress or ingress VL, or any affine combination of both, does not affect the cost of the SFC embedding. VL latency constraints are enforced by disregarding PPs, violating the delay limit. For the example depicted in Figure 5.5, $k = 1$. For $k > 1$, the AG is a multi-graph.

service	run	Cost					Resulting composition	ΔU before run (θ, l_{out})	$\Delta\Theta_{black}$ before run	ΔM_{res} before run
		c_0	c_1	c_2	c_3	c_4				
s_0	0	**0.59**	1.51	1.18	1.22	1.01		(θ_0, l_0)		
s_0	1		1.30	0.97	1.01	**0.81**		(θ_1, l_1), (θ_1, l_2)		(θ_0, b_1), (θ_1, b_1), (θ_2, b_2)
s_0	2				**0.79**			(θ_4, l_3), (θ_4, l_4), (θ_7, l_5)	θ_3	(θ_4, b_3), (θ_7, a_4), (θ_{11}, b_4)
s_0	3						$C_B = \{c_0, c_3, c_4\}$	(θ_6, l_5)		(θ_6, a_2), (θ_{10}, b_2)
s_1	0	**0.90**	2.08	1.98	1.87	1.64		(θ_0, l_0)		
s_1	1		1.81	1.71	1.61	**1.38**		(θ_1, l_1), (θ_1, l_2)		(θ_0, b_1), (θ_1, b_1), (θ_2, b_2)
s_1	2				**1.31**			(θ_4, l_3), (θ_4, l_4), (θ_7, l_5)	θ_3	(θ_4, b_3), (θ_7, a_4), (θ_{11}, b_4)
s_1	3						$C_B = \{c_0, c_3, c_4\}$	(θ_6, l_5)		(θ_6, a_2), (θ_{10}, b_2)

Table 5.5: Illustration of GCS for the placement of services s_0 and s_1, which can each be realized by composition C_A or C_B.

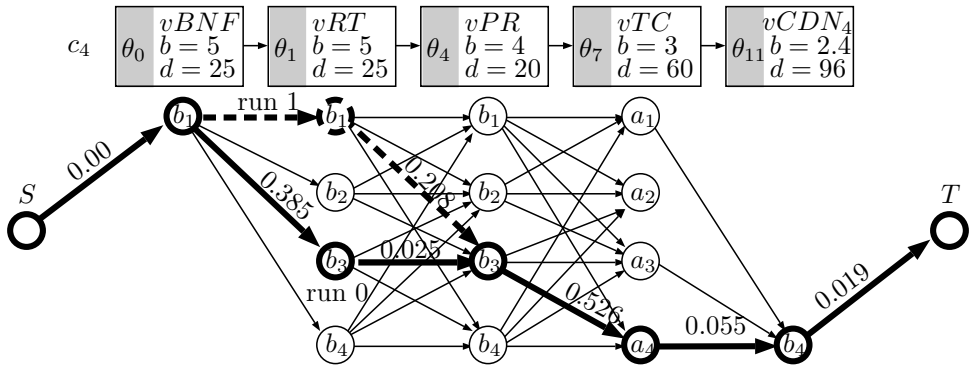


Figure 5.5: AG used to estimate the embedding costs and to generate an embedding of c_4 for s_0. The minimum-cost embedding for run 0 and the deviation from this embedding in run 1 are indicated in solid thick, and solid intermittent lines, respectively.

Compared to the SFC approach by Bari et al. [64], we consider all k ShPs in the SNe, while they consider only a single ShP between candidates PMs in subsequent stages. Additionally, Bari et al. check the resource constraints solely in a pre-processing step, which is only valid if resources can only be used once by a chain, i.e., VNFs of the same request cannot be collocated.

Second, $ME(G_c, \kappa) : c \in C, \kappa \in \mathbb{N}$ generates the κ^{th} minimum-cost embedding by determining the κ^{th} ShP from S to T in G_c. The resulting ShP in the AG corresponds to an embedding. The embedding is not guaranteed to be feasible, as PM- and PL-capabilities, or end-to-end latency bounds might be exceeded. The cost function (Equation 5.16) results in an embedding that tends to avoid highly-loaded resources. If the embedding is not feasible, due to collocation of too many VNFs, then the algorithm is increasingly likely to yield a feasible solution, as κ increases.

For a graph with n nodes and m edges the algorithm proposed by Martins et al. has a worst-case complexity $O(Kn(m + n \log n))$ [113]. The maximum number of PM candidates for any VNF is given by $A = \max_{v \in V}\{|\phi(v)|\}$. For the AG $n = O(A|V|)$ and $m = kA^2$. Hence, generating the K minimum-cost embeddings for a SFC is $O(KA|V|(kA^2 + A|V| \log A|V|)$.

5.5.3 Illustration

In this section, the execution of GCS is illustrated using the working example from Section 5.3. An overview of the state of the algorithm during each iteration is provided in Table 5.5. Initially, U contains the combinations of θ_0 corresponding to v_{root}, with its egress VLs l_0 and l_1, $cover = \emptyset$ and $\Theta_{black} = \emptyset$. Then, during the first iteration the minimum-cost embedding of each candidate chain is determined. The AG used to find the minimum-cost embedding for c_4 of s_0 is shown in Figure 5.5. The network traffic corresponding to c_4 must flow through $vBNF$, vRT, vPR, vTC, and $vCDN_4$, in this specified order, as shown in Figure 5.4.

In run 0, no augmented VNFs are mapped ($M_{res} = \emptyset$), and all candidates in $\phi(\theta(v))$ are considered. Each candidate chain covers exactly one element in U. Therefore, the chain with minimum cost is selected: c_0. Since, each chain can only be selected once, c_0 is not considered in later iterations. U is updated with the combinations of each augmented VNF

in M_{res}, combined with their outgoing links. At the start of iteration 1, the only element in U, not covered by M_{res} is (θ_1, l_2). Since, not all elements are covered, the algorithm cannot finish yet. In iteration 1, the minimum embedding cost of c_1 through c_4 is estimated. Now, the mapping possibilities are more limited as θ_0 and θ_1 were already mapped to b_1 and b_3, respectively. Compared to run 0, the embedding cost for c_1, c_2 and c_3 is lowered because θ_0 and θ_1 are already embedded. Again, each considered chain covers one additional element from U, namely (θ_1, l_2). Therefore, the chain with minimum cost, i.e. c_4, is selected. Each outgoing link of an augmented VNF can only be placed once. Therefore, the addition of θ_4 to the composition, leads to the blacklisting of θ_3 and the chains that contain this augmented VNF, namely c_1 and c_2. At the end of run 1, outgoing VL l_5 of θ_6 remains uncovered. Therefore, the algorithm cannot terminate. In run 2, the only candidate chain remaining, i.e. c_3, is selected. Before iteration 3, all required elements in U are covered and the algorithm terminates successfully. For this particular problem instance, the minimum-cost embedding was feasible. When the minimum-cost embedding of the selected chain is not feasible, then the algorithm tries the next $K-1$ minimum-cost embeddings. If none of these embeddings satisfy the resource constraints, then the algorithm terminates unsuccessfully.

Both s_0 and s_1 are placed using C_B, the corresponding costs are 2.19 and 3.59 respectively. The costs are slightly higher than for OPT-SECC, because GCS does not consider the impact that the embedding of one chain has on the embedding costs during subsequent iterations.

5.6 Performance evaluation

This section compares the proposed algorithms against the state of the art. First, the procedures to generate composition and embedding constraints are described. Then, the evaluated placement algorithms are discussed. Next, the simulation parameters and evaluation metrics are provided. Finally, numerical results conclude this section.

5.6.1 Requirements

Composition requirements Chaining requirements are generated using a procedure that synthesizes random Directed Acyclic Graphs (DAGs), based on node-ranking: each VNF is assigned a rank. Each egress link of a VNF of certain rank depends on all VNFs of higher rank. Dependencies on VNFs of lower or equal rank are not allowed. Rank 1 contains only v_{root}. The highest rank, contains only the terminating VNFs. Based on the number of ranks, we distinguish between two scenarios. Both scenarios result in a minimum of 5 and a maximum of 8 VNFs per chain.

- 3 ranks: the number of VNFs in rank 2, is with equal probability selected from $\{3, 4, 5, 6\}$.
- 5 ranks: the number of VNFs in ranks 2, 3 and 4, are each with equal probability selected from $\{1, 2\}$.

The initial bandwidth $r_{init} = 0.10$. The relative bandwidths $r_{rel}(l)$ on VL $l \in L$ are uniformly distributed in $[0.75; 1.5]$. The relative processing requirements $d_{rel}(v)$ are uniformly distributed in $[1; 4]$. Each VNF has at most τ outgoing links. The out-degree distribution for

VNFs follows a power law, i.e., for each VNF the probability of k outgoing links $P(k) = ak^{-\gamma}$, where a is a normalizing constant such that the $\sum_{k=1}^{\tau} kP(k) = 1$. Based on the maximum out-degree τ, the VNF-FG is either an SFC ($\tau = 1$), or an SFT ($\tau = 5$). In the SFT scenario, $\gamma = 2.8$, resulting in an average out-degree of 1.3. A single-chain configuration results in at most 720 and 8 valid chains for 3 and 5 ranks, respectively.

Embedding constraints Transit-stub Substrate Graphs (SGs) are generated using the GT-ITM topology generator on a 100 × 100 grid [106]. Any two nodes in the transit network are connected by a PL with a probability of 80%. Within a stub-network cluster this probability is 40%. Each PM in the transit network is connected to 2 clusters, each comprising 6 PMs. In the experiments, the number of transit nodes is 2, corresponding to 26 PMs. Processing capabilities are uniformly distributed in [20; 100]. The PL bandwidths all equal 4.

The LCs are generated in the following way, a similar approach is taken in [78], [49]: the VNFs are each assigned a preferred location, which corresponds to the location of a randomly selected PM $h_{pref}(v) \in H$. The candidate set for v is the collection of PMs, located within a distance $radius$ from $h_{pref}(v) \in H$.

$$\phi(v) = \{h \in H | dist(h, h_{pref}(v)) \leq radius\} : \forall v \in V, \tag{5.17}$$

where $dist(h_1, h_2)$ is the Euclidean distance between PMs h_1 and h_2. The coordinates generated by GT-ITM are unitless, consequently the same applies for radius.

Service arrival The arrival of service requests is simulated in a discrete event simulator. The algorithm handles the requests sequentially, in the order of arrival. After each accepted request, the remaining bandwidth and processing capabilities are updated. When the life-time of a service exceeds the requested duration, the consumed resources are released. The time between arrivals and the request holding times are assumed exponentially distributed with rate $\lambda = 1$ and μ, respectively.

5.6.2 Evaluated algorithms

Following algorithms are evaluated.

OPT-SECC the exact algorithm presented in Section 5.4. The ILP is solved using Gurobi 7.5.2, a hybrid solver, combining multiple solution techniques [114].

GCS the heuristic presented in Section 5.5.

MEHR the uncoordinated approach by Mehraghdam et al., performing VNF-FG composition and embedding in two separate steps [86]. First, VNF candidates are chained greedily, in order of increasing bandwidth requirement. For the working example, this means that only C_A is considered. Second, this VNF-FG is embedded using a Mixed Integer Quadratically Constrained Program. For comparison's sake, in our implementation the embedding is performed using the ILP described in Section 5.4.4. The algorithm can only place one of the two services in the working example.

ILP2STAGE an uncoordinated approach, performing VNF-FG composition in two separate stages. First, the VNF-FG is composed by an ILP, minimizing the aggregate VL bandwidth, subject to the constraints in Section 5.4.3. The composition objective is the same as in [66]. Then, this VNF-FG is embedded using the ILP described in Section 5.4.4.

SUBG the heuristic based on subgraph isomorphism detection proposed by Beck et al. [85]. This recursive algorithm performs chaining, node and link embedding at the same time. At each step this DFS algorithm lists the possible VNF candidates to add to the chain. For each candidate VNF, the algorithm generates a list of candidate PMs within the neighborhood of the most recently added PM. When the outgoing VLs of the current VNF cannot be mapped, then the algorithm backtracks. The algorithm places s_0 using composition C_A. Due to the poor load balancing for this request, s_1 cannot be accepted.

5.6.3 Simulation parameters

The key simulation parameters are:

- *Radius*: geographical distance from the preferred PM, where a VNF can be placed. It is a measure for the flexibility in the embedding: a low radius corresponds to a very location-constrained embedding;
- r_{init}: initial data rate arriving at v_{root}. The processing and bandwidth requirements in the VNF-FG are proportional to r_{init}. Therefore, an increased r_{init} corresponds to a higher offered load;
- *Ranks*: the number of ranks used to generate the precedence requirements. For a given number of VNFs, fewer ranks correspond to fewer precedence constraints, increasing the chaining flexibility; and
- *VNF-FG topology*: either an SFC or an SFT.

5.6.4 Evaluation metrics

Following metrics are evaluated. First, the *Acceptance ratio* is the fraction of requests that is accepted. Further, the total *bandwidth* and *CPU* consumption in the SNe, combined for all requests are evaluated. Finally, the required *Computation time* to process a request is evaluated.

5.6.5 Results

Two scenarios are considered. In the *offline* scenario, a batch of 100 requests is processed. For each batch, the evaluated metrics are averaged out over these 100 requests. The holding time $\mu \to \infty$, meaning that all requests are active at the same time. Per parameter setting, 10 independent batches are averaged out. In the *online* scenario, the holding time $\mu = 100$. Therefore, in steady-state the expected number of requests active at the same time is 100. The resulting traces are smoothed by a moving average filter with a window-size of 100 requests.

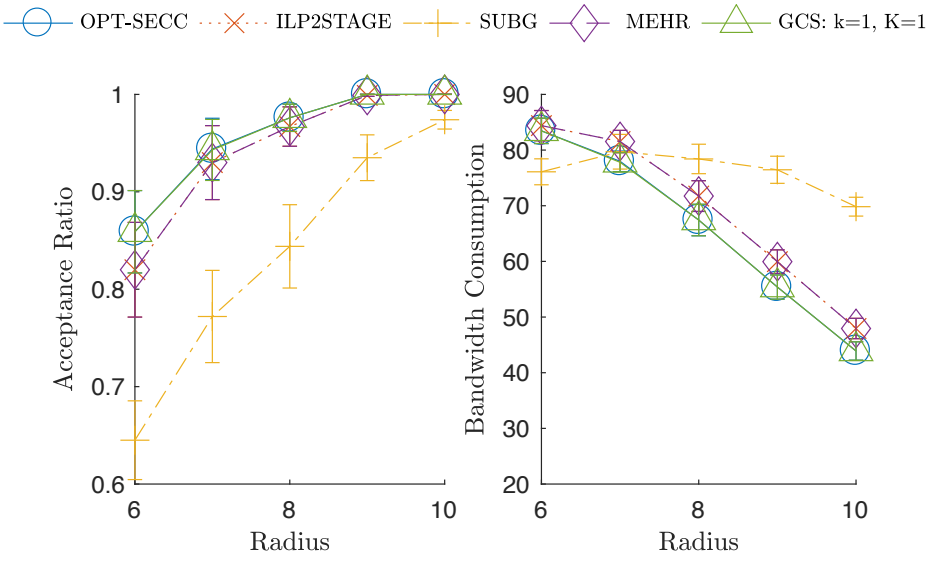
Service Function Chain VNF-FG

Offline Figure 5.6 shows the impact of the LCs for 5 VNF ranks. A valid chaining contains all VNFs in V. The cardinality of V is between 5 and 8. Overall, the acceptance ratio of OPT-SECC is the highest, closely followed by GCS, ILP2STAGE and MEHR. The difference in acceptance ratio between OPT-SECC and MEHR is limited to 4%. The acceptance ratio of ILP2STAGE equals that of MEHR. The acceptance ratio of SUBG is up to 22% lower. The computation time of ILP2STAGE is the highest, followed closely by OPT-SECC and MEHR. In this scenario, the performance gain obtained by coordinating composition and embedding is limited because the maximum number of valid chains is only 8. The computation time for SUBG and GCS is about 100× lower, compared to OPT-SECC. For a radius of 6, only a few PMs can host a particular VNF. The acceptance ratio is at its lowest or all algorithms. As the radius increases, the cardinality of each candidate set goes up. The main reason why SUBG performs worse than the other algorithms is that the procedure terminates as soon as a valid embedding is found, which is not necessarily one with a low-cost VNF-FG. This behavior ultimately limits the acceptance ratio for the next requests. Finally, for higher radii, the acceptance ratio approaches 100% for all algorithms. Clearly, OPT-SECC results in the placement with the lowest bandwidth consumption per accepted request, closely followed by GCS. The placement performance of GCS is virtually equal to that of OPT-SECC, while reducing computation time by a factor of 100. The bandwidth consumptions by MEHR and ILP2STAGE are up to 9% higher than for OPT-SECC, because the composition and embedding stages are not coordinated. The bandwidth consumption of SUBG is up to 60% higher than for OPT-SECC. The main reason for SUBG's poor performance is that the embedding lacks a look-ahead mechanism to consider the placement restrictions on the next VNFs.

Figure 5.7 shows the same results for service requests with 3 VNF ranks, instead of 5, and the same maximum number of VNFs. Obviously, the difference in acceptance ratio between our proposed algorithms and the other algorithms is much larger now, up to 29% for low radii. Relative to Figure 5.6, the acceptance ratios of MEHR and ILP2STAGE remain virtually unaltered. While the bandwidth consumption by GCS and OPT-SECC decreases by up to 38%, for MEHR and ILP2STAGE the decrease is less than 11%.

The acceptance ratio of SUBG increases by up to 5%, indicating that the algorithm can sometimes find a feasible embedding for an alternative composition, when the minimum-bandwidth VNF-FG cannot be embedded. Finally, our proposed algorithms see an acceptance ratio improvement up to 13%, compared to the case with 5 ranks. Interestingly, for our proposed algorithms the processing consumption *decreases* slightly as the radius increases and the acceptance ratio remains constant. The reason for this decrease is that the CPU consumption is proportional to the total VL bandwidth in the VNF-FG. Hence, when LCs are stringent, then our proposed algorithms generate VNF-FGs that match the SNe topology best. When LCs are loosened, then our algorithms generate VNF-FG with reduced bandwidth requirements.

Table 5.6 shows the influence of the SNe dimensions; the number of transit nodes is varied from 1 to 4. As the SNe dimensions increase, the total available processing and bandwidth resources increase. For all algorithms, SUBG excluded, the acceptance ratio increases as the SNe dimensions go up. Since SUBG consumes the most bandwidth resources, the bandwidth



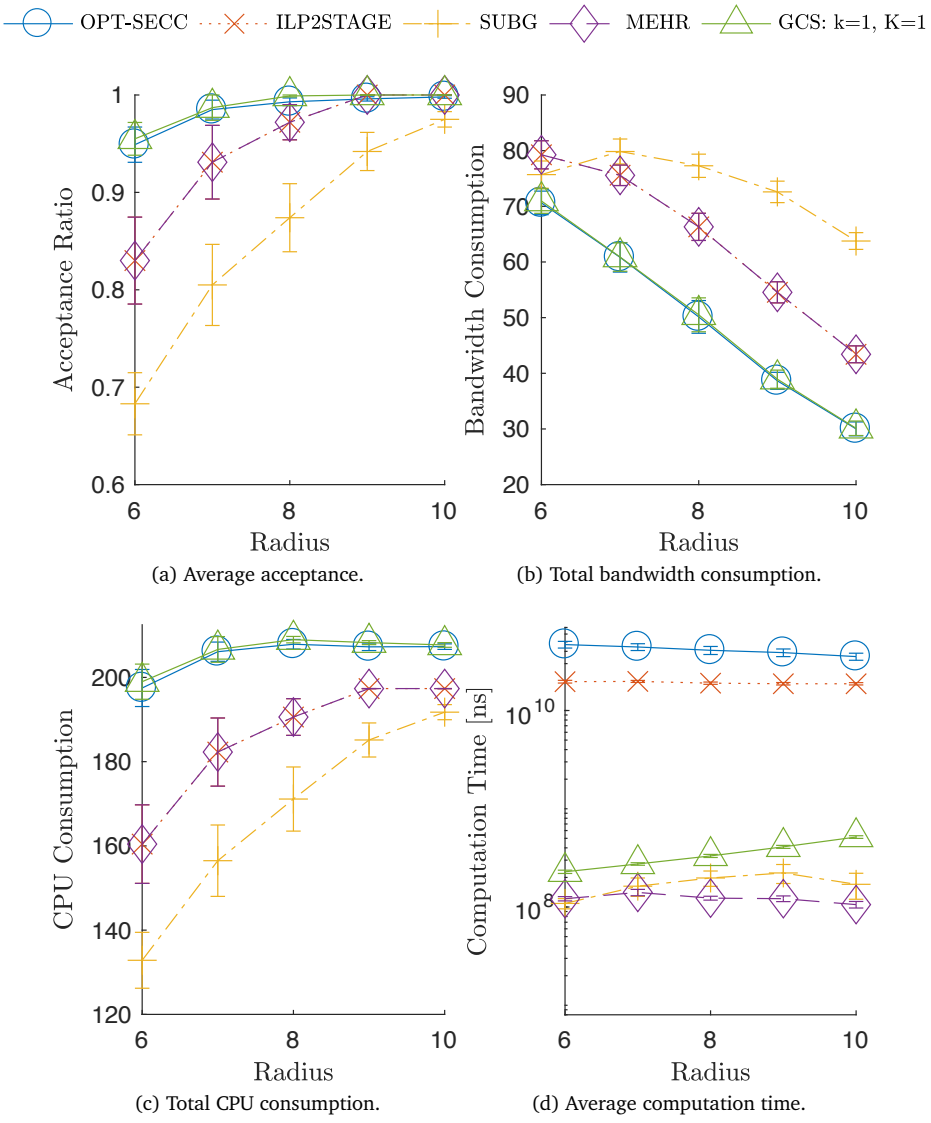
(a) Average acceptance.

(b) Total bandwidth consumption.

(c) Total CPU consumption.

(d) Average computation time.

Figure 5.6: Influence of the LCs: SFC, offline, for 5 ranks.



(a) Average acceptance.

(b) Total bandwidth consumption.

(c) Total CPU consumption.

(d) Average computation time.

Figure 5.7: Influence of the LCs: SFC, offline, for 3 ranks.

PMs	OPT-S.	ILP2.	SUBG	MEHR	GCS
13	0.90, 15	0.82, 6	0.75, 0.03	0.82, 0.09	0.90, 0.15
26	0.95, 49	0.83, 20	0.69, 0.11	0.83, 0.13	0.95, 0.23
39	0.97, 75	0.85, 30	0.68, 0.25	0.85, 0.17	0.97, 0.27
52	1.00, 103	0.95, 46	0.83, 0.47	0.95, 0.42	1.00, 0.30

Table 5.6: Influence of the number of PMs on the acceptance ratio and computation time in seconds for $radius = 6$.

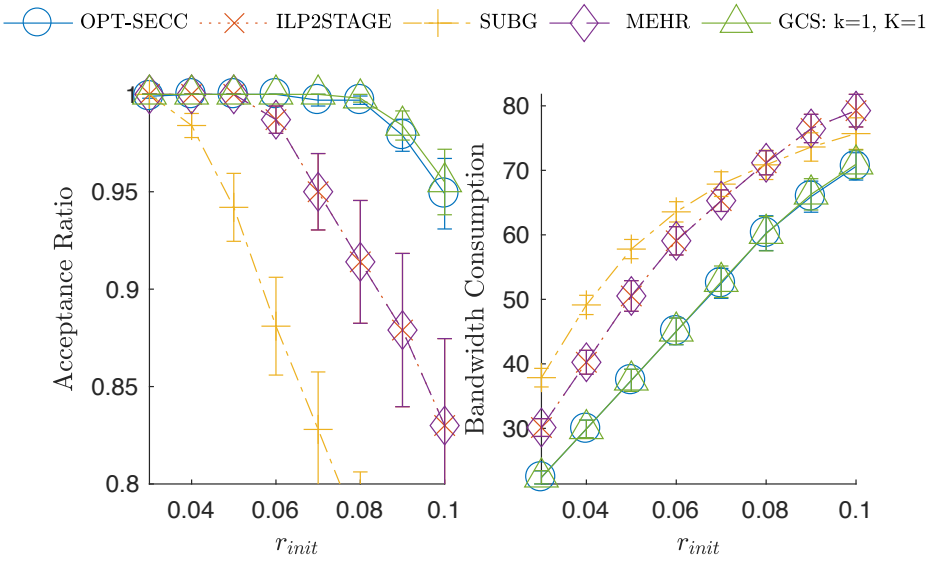
between the transit nodes becomes a bottleneck for 26 and 39 PMs. As the number of transit nodes further increases, the impact of this bottleneck diminishes.

Figure 5.8 shows the impact of the offered load. For low values of r_{init} all five algorithms can handle all requests. In general, when the offered load increases, the acceptance ratio decreases. The acceptance ratio is the highest for OPT-SECC and GCS. For OPT-SECC, the reduced acceptance ratio for higher r_{init} values is due to imposition of a time-limit of 10 minutes per request, after which the request is declined. Again, the placement ratio for ILP2STAGE and MEHR are identical. The placement ratio for SUBG is the lowest. OPT-SECC and GCS use the fewest bandwidth resources, but the most processing resources. The computation time for OPT-SECC is up to 2.4× higher than for ILP2STAGE. Further, the processing time of GCS, MEHR and SUBG is about 100× lower than for ILP2STAGE and OPT-SECC. The processing time for SUBG goes up sharply with increased offered load, as the number of recursions needed to find a valid mapping increases. Again, MEHR provides the same placement quality as ILP2STAGE, while requiring up to 100× less computation time. We conclude that for a two-stage approach, with minimization of the VL bandwidth in the composition stage, greedy composition is a great choice.

Online The online traces are shown in Figure 5.9. Initially, no resources are allocated. For all four algorithms, the first requests can be accepted. Clearly, the acceptance ratio decreases first for SUBG, as it performs the worst load balancing. Overall, the acceptance ratios for OPT-SECC and GCS are the highest. The computation time for SUBG is very low for the first requests, as finding a feasible solution is easy. As the SNe becomes more loaded, this recursive algorithm must perform more backtracking operations, resulting in spikes in computation time. The computation time for our proposed heuristic is fairly constant over time.

Service Function Tree VNF-FG

Offline The influence of LCs on the embedding of an SFT are shown in Figure 5.10. Again, OPT-SECC outperforms the other algorithms in terms of acceptance ratio. For radius levels < 6 (not visible), the difference in acceptance ratio between OPT-SECC and the algorithm based on GCS is significantly larger than for the SFC scenario. The reason for this difference is twofold. First, GCS does not consider that the current chain selection influences the universe of elements that need to be covered in the next iterations. The only exception is when only the root VNF has a degree > 1. Second, the embedding of one chain influences the costs of chains



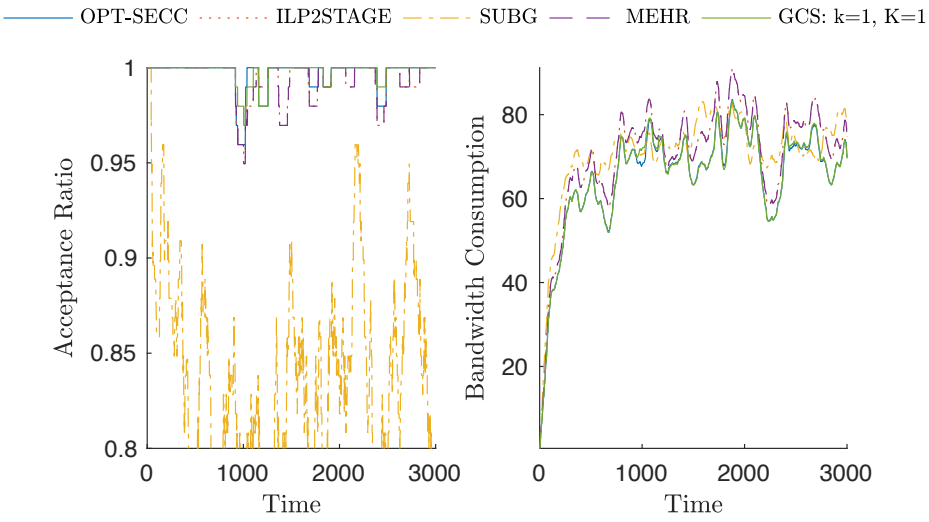
(a) Average acceptance.

(b) Total bandwidth consumption.

(c) Total CPU consumption.

(d) Average computation time.

Figure 5.8: Influence of the offered load: SFC, offline, for 3 ranks.



(a) Acceptance.

(b) Total bandwidth consumption.

(c) Total CPU consumption.

(d) Computation time.

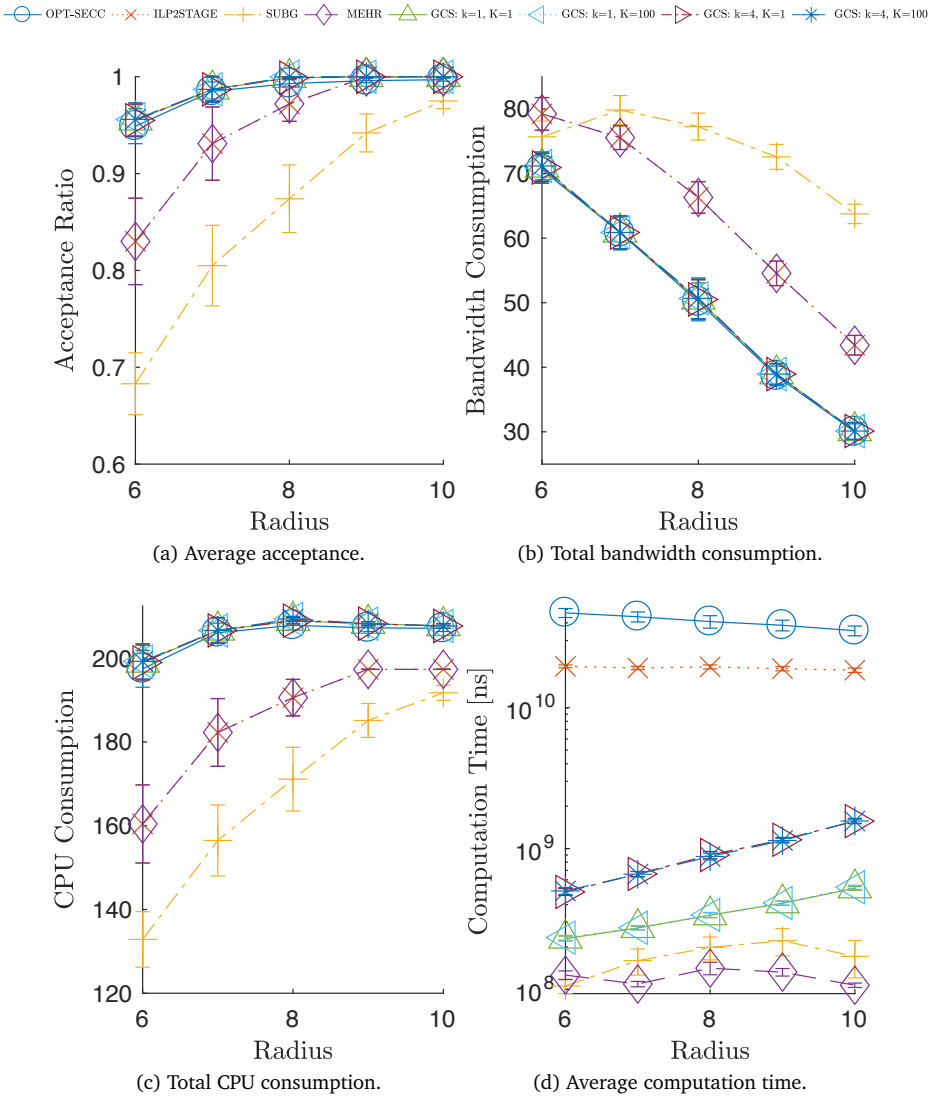
Figure 5.9: SFC, online traces, for 5 ranks, $radius = 9$.

which share the same augmented VNFs. An unfortunate placement of an augmented VNF in one iteration can lead to an increased bandwidth cost during subsequent iterations. Again, the acceptance ratio for SUBG is the lowest because of inefficient bandwidth consumption. GCS is up to 25× faster than OPT-SECC. For GCS, the number of PPs considered between any two PMs (k), has a minor impact on bandwidth consumption. For $k > 1$, the algorithm can prefer a lower loaded path over the shortest path, increasing the consumed bandwidth slightly. In a more connected SNe, k is expected to have a higher impact. When the radius, and consequently, the cardinalities of the candidate sets are sufficiently large, the computation time is proportional to k. The number of lowest cost embeddings to consider for a chain (K), only has a minor impact on the acceptance ratio. For $K = 1$ and $K = 100$, the computation times are virtually identical, meaning that only rarely additional embeddings are explored. Often, alternative embeddings for a given chain are limited, when many augmented VNFs along this chain were already mapped in previous iterations. The impact of K will be larger for more stringent processing capability constraints.

Online The online traces for an SFT are shown in Figure 5.11. Compared to the SFC scenario, the computation times display a larger variance, as they strongly depend on the out-degree of the VNFs. Again, initially all algorithms can place the incoming requests. Then, the acceptance ratio drops rapidly for SUBG, due to inefficient bandwidth use. The acceptance ratio of OPT-SECC is the highest, closely followed by GCS, and MEHR. For GCS, the average acceptance rate for $K = 1$ and $K = 100$ are almost equal. Given a particular request and SNe-loading, $K = 100$ is more likely be successful. However, $K = 1$ tends to place only *cheaper* requests. Both effects seem to cancel each other out.

5.6.6 Conclusions

In order to make on-demand service composition and embedding an integral part of the future Internet architecture, efficient placement algorithms are required. In this subsection, we proposed algorithms to find an initial placement for service requests that arrive sequentially. These novel algorithms differ from previous algorithms, in that they have better coordination between the composition and embedding stages. Coordination between these stages can greatly improve the quality of the placement but increases the solution space greatly. To this end, we first formulated the problem as an ILP on an augmented VNF tree with precomputed chains, satisfying the precedence requirements. Then, we proposed a fast heuristic based on greedy chain selection and embedding. The improved load balancing and reduced hosting costs in our algorithms outperformed the existing approaches in terms of acceptance and provisioning cost, as shown through simulation.



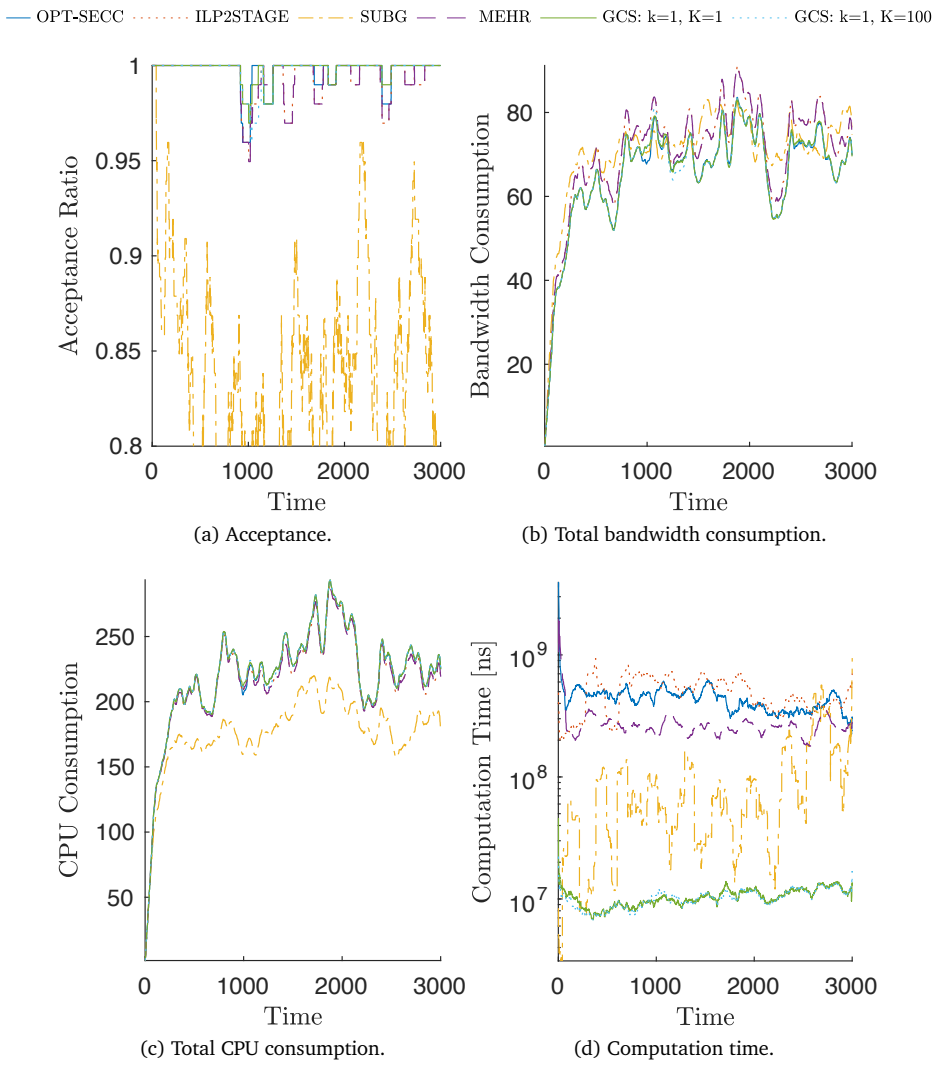
(a) Average acceptance.

(b) Total bandwidth consumption.

(c) Total CPU consumption.

(d) Average computation time.

Figure 5.10: Influence of LCs: SFT, offline, for 3 ranks.



(a) Acceptance.

(b) Total bandwidth consumption.

(c) Total CPU consumption.

(d) Computation time.

Figure 5.11: SFT, online traces, for 5 ranks, $radius = 9$.

Chapter 6

An improved NFV orchestration model

This work was published in [C7]. An extended version is currently under review [C3].

Similar to Chapter 5, this chapter is dedicated to the orchestration of NSs in a geo-distributed NFV environment. This challenge is captured by Question III in Chapter 1. Again, we investigate how NSs can be orchestrated based on their SRs and how the coordination of the composition and embedding subtasks can mitigate the issues arising from the heterogeneity in the infrastructure. The major limitations of the previous chapter and related NFV orchestration approaches were in the service model, i.e. the VNF-FG topology was assumed a tree. While the content of traditional IT services was often managed centrally, many novel Internet services require traffic aggregation, as discussed in Chapter 1. Therefore, this chapter investigates how the SRs of NSs that require traffic aggregation can be modeled, and how these services can be orchestrated in geo-distributed NFV environments.

Recently, network services are increasingly connecting computational elements within and across datacenters. In the NFV environment, to successfully orchestrate a network service, first a VNF-FG must be composed that realizes the required functionality. Second, this VNF-FG must be embedded onto the infrastructure, that is increasingly becoming heterogeneous. To efficiently allocate service demands, intelligent mechanisms and algorithms are needed to effectively tailor the VNF-FG to the cloud network onto which the service will be deployed. This chapter introduces the first service model supporting on-demand orchestration of services with bidirectional chaining and delay constraints, optional VNFs and traffic aggregation. Building on this service model, we propose algorithms looking for the optimization of the order and number of VNF instances, to adapt the VNF-FG to the available resources in the substrate network. Numerical experiments show that, through coordination of composition and embedding tasks, our proposed algorithms can significantly improve the acceptance ratio, compared to algorithms that perform these tasks in two separate stages.

6.1 Introduction

In this chapter, we propose a more widely applicable service model that can produce VNF-FGs of the DAG class, meaning that the traffic originating from multiple VNFs can be aggregated in a single VNF instance. Further, our model supports more general chaining requirements and optional performance enhancing VNFs. Through our proposed optimal algorithm that performs both composition and embedding in one single stage, we demonstrate the importance of considering the SNe during composition. This consideration of the SNe is especially important when optional VNFs are involved that can either ease or complicate the embedding. To solve the scalability problems related to the optimal solution of the problem, we present a fast heuristic that can be used to orchestrate real-life services. This recursive heuristic, based on subgraph isomorphism detection, extends a given PC by adding VNF and VL instances to the VNF-FG and embedding them at the same time.

The structure of this chapter is the following. First, the related work is discussed in Section 6.2. Subsequently, the combined composition and embedding problem is formulated as an ILP in Section 6.3. Then, to deal with the computational complexity associated with finding an exact solution, a fast heuristic based on subgraph isomorphism detection is proposed in Section 6.4. Next, the proposed algorithms are evaluated in Section 6.5. Finally, Section 6.5.6 concludes this chapter.

6.2 Related work

In a recent survey, the resource allocation problem in NFV (NFV-RA) has been divided in three main challenges [59]. 1) The composition challenge asks the following question: How to concatenate the different VNFs efficiently in order to compose a NS in the most adequate way, with respect to the operator goals?; 2) The embedding challenge that, after the VNFs are composed, seeks to find where to efficiently allocate the VNFs in the network infrastructure accomplishing the QoS service constraints; and 3) the SFC scheduling that seeks to answer the question: How to execute each function in order to minimize the total execution time without degrading the service performance and respecting all the precedences and dependencies between the VNFs composing the NS?.

Most of the current work has been devoted to solve the second stage of the problem that is a specific variant of the VNE problem [47]. It is well-known that the VNE problem is NP-hard [48]; hence, a wide range of optimal and heuristic algorithms have been proposed in the literature to solve it. Generally, these approaches consider processing and memory limitations of the PMs onto which the virtual nodes are located and bandwidth limitations of the PLs over which the VLs are routed. Further, authors have considered additional QoS constraints that are important to practical applications. One important QoS constraint, especially for 5G services, is the delay that a service can tolerate. For instance, the Small Cell Forum lists the maximum allowed delay for the communication between C-RAN VNFs [36]. Hence, some researchers consider a maximum delay on VL instances between VNFs [115]. In contrast, Luizelli et al. consider a maximum end-to-end delay while embedding a SFC [82]. The investigators compute the end-to-end delay of a SFC as the sum of the delays and VNF processing times. While Luizelli et al. consider solely the propagation along a path in the SNe, other researchers

model the impact of workload variations on delay. For instance, Innoue et al. propose a distributed approach to the VNE problem, that minimizes the migrations caused by uncertainty in the delays using the Yuragi method. The authors model the VNF processing and forwarding delays by applying queuing theory [75]. The calculated delay is based on the loading of the underlying physical resources, i.e. hosting PM and single highest-loaded PL on the PP, respectively.

While the aforementioned approaches do not consider the composition stage of the problem, there are some recent proposals that face this problem. Authors of [66] were the first to propose an exact ILP-based approach to solve the composition problem, even if this solution is not scalable it can be taken as a first approximation to solve this stage. The first proposal to solve both the composition and embedding stages was proposed in [86], here authors present an uncoordinated solution where the composition is solved via a greedy heuristic and the resulting SFC is embedded using an MIQCP.

The major limitations of [66, 86, 85] and Chapter 5 are in the composition model.

Compared to previous work, this chapter present the following novelties:

- an extended service model, including traffic aggregation and bidirectional chaining requirements, optional VLs and support for both VL and end-to-end delay constraints;
- three new algorithms that can place requests which conform to this new service model; and
- a detailed comparative study to the performance of the presented algorithms.

One algorithm is an optimal algorithm that coordinates both tasks, which can provide the optimal solution when the provider is allowed to compose or tailor the VNF-FG. The other is an algorithm that solves both tasks individually to optimality, in two separate stages. When the provider is provided a precomposed VNF-FG that must be embedded, then the performance will be comparable to this two-stage algorithm. The last proposal is an improved recursive heuristic that can find a solution fast, extended to include latency constraints.

6.3 Problem formulation

This section formally introduces the problem studied in this work. The problem is to find a good quality initial PC, given SRs and the SNe description. First, the composition and embedding requirements are introduced. Second, an AG is introduced to simplify formulation of the problem as an ILP. Third, four closely related problems are formulated as ILPs. The combined composition and embedding problem is formulated as a 1-stage ILP that optimizes the service composition and embedding at the same time. The corresponding formulations without and with delay-constraints are provided in Section 6.3.3 and 6.3.5, respectively. Additionally, ILP formulations are provided that can be used to perform VNF-FG composition and embedding in two separate stages. The corresponding formulations without and with delay-constraints are provided in Section 6.3.4 and 6.3.6, respectively. Throughout this work, an illustrative use-case with limited dimensions is used to illustrate the concepts.

Symbol	Description
V	Set of VNFs in the service.
L	Set of VLs in the service.
$L_{out} \subset L$	Set of egress VLs in the service.
$L_{in} \subset L$	Set of ingress VLs in the service.
$L_{out}^{v} \subset L_{out}$	Set of egress VLs of $v \in V$.
$L_{in}^{v} \subset L_{in}$	Set of ingress VLs to $v \in V$.
$V_{init} \subset V$	Initial VNFs where the service originates.
$V_{term} \subset V$	Set of terminating VNFs.
$r_{init}(v) : V_{init} \rightarrow \mathbb{R}^{+}$	Initial data rate arriving at initial VNF $v \in V_{init}$.
$\mathscr{C}(l_{out}, l_{in}) : L_{out} \times L_{in} \rightarrow \{0, 1\}$	Binary parameter indicating if l_{in} and l_{out} are compatible.
$M(l_{out}) : L_{out} \rightarrow \mathbb{Z}^{+}$	Maximum number of VL instances flowing out of a VNF instance via l_{out}.
$L_{l_{out}}^{next} \subset L_{in}$	Set of ingress VLs that the traffic on $l_{out} \in L_{out}$ should pass through next.
$L_{l_{in}}^{prev} \subset L_{out}$	Set of egress VLs that the traffic on $l_{in} \in L_{in}$ should have passed through previously.
$d_{rel}(v) : V \rightarrow \mathbb{R}^{+}$	The ratio of the processing requirement of VNF v, to the ingress data rate at v.
$r_{rel}(l_{out}) : L_{out} \rightarrow \mathbb{R}^{+}$	The ratio of traffic flowing out $l_{out} \in L_{out}$, to the traffic generation rate of VNF $v \in V$.

Table 6.1: Input parameters to the service composition.

6.3.1 Composition and embedding requirements

The notation used for the SRs is described in Table 6.1.

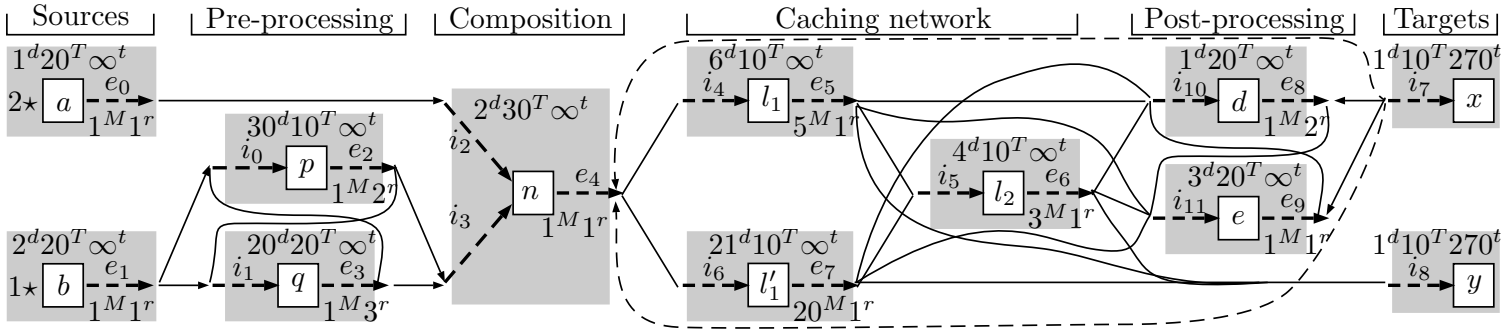


Figure 6.1: Service requirements for the illustrative use-case, grouped per VNF in gray rectangles. Arrows with intermittent lines whose label start with an i and an e, represent ingress and egress VLs respectively. A solid line connecting $l_{out} \in L_{out}$ and $l_{in} \in L_{in}$ indicates $\mathscr{C}_{l_{out},l_{in}} = 1$. An arrowhead on this same line from l_{out} to l_{in} indicates that $l_{in} \in L^{next}_{l_{out}}$. An opposite arrowhead indicates that $l_{out} \in L^{prev}_{l_{in}}$. For all $v \in V$, $r_{init}(v)^{\star}$, $d_{rel}(v)^{\mathbf{d}}$, $T^N(v)^{\mathbf{T}}$ and $t^N(v)^{\mathbf{t}}$ are indicated. Further, for all $l_{out} \in L_{out}$: $M(l_{out})^{\mathbf{M}}$ and $r_{rel}(l_{out})^{\mathbf{r}}$ are indicated and $t^L(l_{out})$ =120 ms.

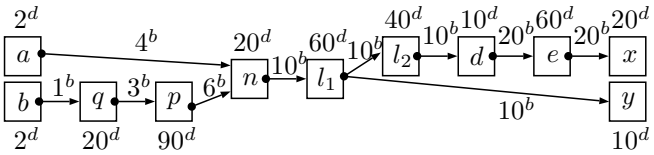
The NS's functionality is realized by a VNF-FG. In this section, the VNF-FG is considered a DAG. This VNF-FG comprises VNF instances, interconnected by VL instances. A path in this DAG, formed by an ordered set of VL instances, is referred to as an SFC. The VNF instances in the VNF-FG are created based on a set of VNFs (V). Each VNF $v \in V$ has a set of egress VLs (L^v_{out}) and ingress VLs (L^v_{in}). The processing requirements of an instance of $v \in V$ are proportional to the total ingress bandwidth to the instance, by a factor $d_{rel}(v)$.

An instance of VNF $v_1 \in V$ can communicate to an instance of VNF $v_2 \in V$ through a VL instance, that is formed by connecting an egress VL of the source VNF instance $l_{out} \in L^{v_1}_{out}$ to an ingress VL of the target VNF instance $l_{in} \in L^{v_2}_{in}$. VL l_{out} can be connected to l_{in} i.f.f. both VLs are compatible, i.e. $\mathscr{C}(l_{out}, l_{in}) = 1$.

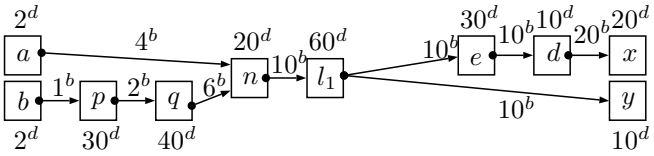
The VNF-FG must contain exactly one VNF instance of each initial VNF $v_{init} \in V_{init}$ and terminal VNF $v_{term} \in V_{term}$. An initial VNF $v_{init} \in V_{init}$ has no ingress VLs and generates traffic at a rate $r_{init}(v_{init})$. For an instance of a non-initial VNF $v \in V \setminus V_{init}$, the bandwidth on each VL instance corresponding to egress VL $l^v_{out} \in L^v_{out}$ is proportional, by a factor $r_{gen}(l_{out})$, to the total ingress bandwidth to this VNF instance. A VNF is terminal i.f.f. it does not have any egress VLs.

In the VNF-FG, the chaining requirements for an instance v^{inst} of VNF $v \in V$ are the following. On the one hand, there are requirements related to the neighborhood of v^{inst}. First, through each of its ingress VLs $l_{in} \in L^v_{in}$, v^{inst} must be connected to one parent. Second, v^{inst} can be connected to at most $M(l_{out})$ children, through each of its egress VLs $l_{out} \in L^v_{out}$. On the other hand, there are bidirectional chaining requirements related to v^{inst}. First, for any terminal VNF instance that can be reached from v^{inst} through its egress VL $l_{out} \in L^v_{out}$, any SFC to this descendant must contain all ingress VLs in L^{next}_v. The set of required ingress VLs that any SFC from v^{inst} to any terminal VNF instance must contain, will be denoted as $N(v^{inst}) \subset L^{next}$. Then, for any child $\tilde{v}^{inst}$ of v^{inst} that is connected via l_{out} to l_{in}, $N(\tilde{v}^{inst}) = N(v^{inst}) \cup L^{next}_{l_{out}} \setminus \{l_{in}\}$. In other words, the ingress VLs that the traffic originating from a child VNF instance must pass through prior to termination, are the required ingress VLs of its parents, plus the required ingress VLs of l_{out}, minus l_{out}. VL l_{out} is removed because this dependency is potentially satisfied. Since a terminal VNF has no egress VLs, each of its instances ${v_{term}}^{inst}$ must have resolved all dependencies of its ancestors, i.e. $N({v_{term}}^{inst}) = \emptyset$. Egress VLs that are not in L^{prev} are referred to as *optional*. It is assumed that optional VLs do not change the bandwidth requirements along an SFC, otherwise the level of service would depend on the VNF-FG composition. Further, for any initial VNF instance ${v_{init}}^{inst}$ that can reach v^{inst} through an SFC that contains $l_{in} \in L^v_{in}$, this SFC must contain all egress VLs in $L^{prev}_{l_{in}}$. The set of required egress VLs that any SFC, from any initial VNF instance towards v^{inst} contains, is denoted as $P(v^{inst}) \subset L^{prev}$. This set determines which children can be connected to v^{inst}.

For the use-case, the chaining requirements are shown in Figure 6.1. This NS makes a composition (VNF n) of two sources (VNFs a and b). VNF n aggregates the traffic flowing onto its ingress VLs, i.e., i_2, i_3. In contrast, aggregation of traffic is not supported in [C2], [66, 85]. These ingress VLs each need to be connected to the egress VL of a and b respectively. Since e_0 and i_2 are interconnected, i.e. $\mathscr{C}(e_0, i_2) = 1$, an instance of a and n can be interconnected via a VL instance formed by connecting e_0 to i_2. In contrast, $\mathscr{C}(e_1, i_3) = 0$ and b and n cannot be directly interconnected; the flow on e_1 must first flow through e_2 and e_3 of pre-processing VNFs p and q. The VNF-FG can only comprise a single instance of each initial VNF, and $M(e_0) = 1$.



(a) VNF-FG$_1$: a composition that chains p after q and e after d.



(b) VNF-FG$_2$: minimal bandwidth composition, chaining p after q and e after d.

Figure 6.2: Illustration of a valid VNF-FG for the use-case with annotation of processing^d and bandwidth^b requirements.

Thus, the service can contain only a single instance of n. The terminal VNFs, i.e., x and y, each require a copy of the composed NS from n. Additionally, these terminal VNFs can have their own specific service delivery requirements. For instance, the ingress VL to x requires the traffic on its ingress VL, i.e. i_7, to have passed through the egress VL of post-processing VNF d and e first. The sequence in which this stream passes through d and e does not matter for the functionality of the service. Since $M(e_8) = 1$, only a single stream can be processed by any instance of d and e. VL i_7 can be directly connected to e_8 or e_9 (solid line), but cannot be directly connected to e_4 (dotted line). While the ingress VL to x does require VNFs d and e, the VL ingress to y, i.e. i_8, does not. Moreover, since i_8 is not compatible with e_8 and e_9, an SFC arriving at i_8 cannot have flown through d nor e.

The distribution of the composed NS from n towards the post-processing and target VNFs is done by a hierarchical caching network. The first layer contains either l_1 or l_1', neither one's inclusion in the VNF-FG is strictly required. Caches l_1 and l_1' can serve post-processing VNFs and instances of layer-two cache l_2. VL l_2 can serve up to 3 VNFs and is optional. While our model supports bidirectional chaining requirements and optional VLs, [66, 85], [C2] consider only chaining requirements from the initial VNF towards the terminal VNFs and mandatory VLs.

Figure 6.2 shows two VNF-FG satisfying the chaining requirements of the use-case.

The VNF-FG in Figure 6.2a chains p after q and e after d. The caching hierarchy comprises one instance of l_1 and l_2 each. The VNF-FG in Figure 6.2b chains q after p and d after e. This composition has the minimum VL bandwidth. The caching hierarchy comprises one instance of l_1. In any composition, the total ingress bandwidth to composition instance (n) equals $4 + 6 = 10$. Since $r_{rel}(e_4) = 1$, in both compositions, the egress bandwidth along the VL instance from n to l_1 equals 10.

The input parameters to the service embedding are listed in Table 6.3. The SNe comprises a set of PMs (H), interconnected by directed PLs (E). The set of PMs on which a particular VNF $v \in V$ can be executed is given by $\Phi(v)$. The remaining processing capability of $h \in H$ is

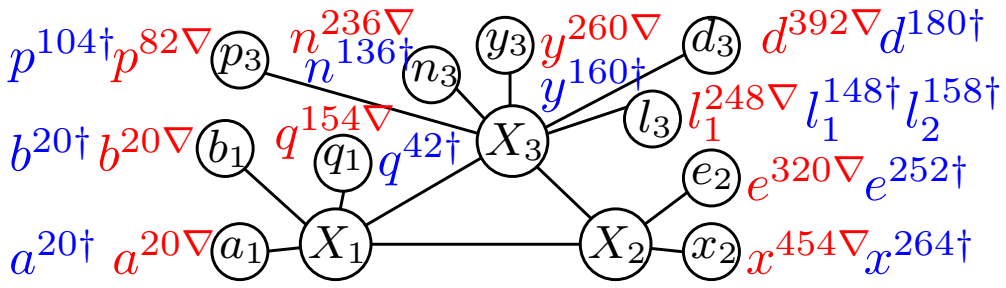


Figure 6.3: SN illustration comprising 3 switches X_1, X_2, X_3. PLs between switches introduce 50 ms of delay. The other PLs introduce 1 ms. Bandwidth and processing capabilities all equal 100. VNF placement and corresponding delays of PCs using the VNF-FG of Figure 6.2a† and Figure 6.2b$^{\nabla}$ are annotated.

Symbol	Description
H	Set of PMs.
E	Set of PLs.
$\phi(v) \subset H$	Set of PMs $\in H$ that can host VNF $v \in V$.
$D(h)$	Remaining processing capability of PM $h \in H$.
$B(h_1, h_2)$	Remaining bandwidth capability of $(h_1, h_2) \in E$.
$T^L(h_1, h_2)$	Delay along $(h_1, h_2) \in E$.
$T^N(v, b)$	Processing delay of executing an instance of VNF $v \in V$ with ingress bandwidth b.

Table 6.3: Input parameters to the service embedding.

given by $D(h)$. The remaining bandwidth capacity of $(h_1, h_2) \in E$ is given by $B(h_1, h_2)$. The end-to-end delay model builds on the one used in [82]. While there, the authors compute the maximum-end-delay in a single SFC, we perform this calculation for a DAG VNF-FG.

It is assumed that the end-to-end delay formed by any SFC in the VNF-FG from an initial VNF instance towards an instance of $v \in V$ must not exceed $t^N(v)$. Further, the delay of a VL instance cannot exceed the maximum delay of its corresponding ingress VL $l_{in} \in L_{in}$. The propagation delay introduced by PL $(h_1, h_2) \in E$ is given by $T^L(h_1, h_2)$. The delay introduced when executing an instance of VNF $v \in V$ with an ingress bandwidth of b is given by $T^N(v, b)$.

6.3.2 Augmented Graph

The VNF-FGs in Figure 6.2a and 6.2b comprise a single instance of VNF n each. The SFC that arrives at n with source b first passes through q in Figure 6.2a and first passes through p in Figure 6.2b. However, these two instances of n have the same ingress bandwidth; the SFCs arriving at their ingress VLs have passed through the same required VLs; and their egress VLs have the same chaining requirements towards the terminal VNFs. These two instances of n will be called *equivalent*. Given the SRs in Section 6.3.1, two VNF instances, v_1^{inst} and v_2^{inst} are equivalent i.f.f. they (1) correspond to the same VNF $v \in V$; (2) have identical ingress VL dependencies (N) after the VNF instance and the same required egress VLs (P) appear before the VNF instance; and (3) they have the same bandwidth and processing requirements. This equivalence relation is denoted as $v_1^{inst} \sim v_2^{inst}$. Since optional VNFs are assumed to not change the bandwidth requirements and the bandwidth requirements are multiplicative; condition (3) follows from (1) and (2). This property will be used in the AG, which serves as a basis to describe any VNF-FG satisfying the SRs. This AG $G(\Theta, \Psi)$, comprises augmented VNFs Θ, interconnected by augmented VLs Ψ, to represent VNF instances and their possible interconnections. By construction, these augmented nodes and augmented VLs satisfy the bidirectional chaining requirements.

The procedure to generate the AG comprises four steps; its procedure is described in Algorithm 6.1. First, the augmented VNFs corresponding to the initial VNFs are generated (Line 5) and added to Θ (Line 6). For these initial augmented VNFs, both N and P are empty. Then, these augmented VNFs are added to queue q_1 (Line 7). For v_{init} the ingress bandwidth is $r_{init}(v)$, this value is stored in map b (Line 8). In the second step, the egress VLs of the augmented VNFs in q_1 are explored. When an egress VL l_{out} of θ_{cur} in q_1 can be connected to a VNF's ingress VL l_{in}, then a new augmented VNF instance, i.e., θ_{new}, is created (Line 17). This connection can be made i.f.f. l_{out} and l_{in} are compatible; and $P(\theta_{cur}) \cup l_{out}$ contains all egress VLs in $L_{l_{in}}^{prev}$. Augmented VNF θ_{new} is added to Θ and its ingress augmented VL is added to Ψ (Line 18). Augmented VNF θ_{new} is added to q_1 (Line 19) as its egress VLs must be explored, and to q_2 (Line 20), to be processed in the third step. This third step prunes invalid augmented VNFs from the AG (Line 27). There can be two types of invalid augmented VNFs in the AG. The first type is terminal augmented VNFs with remaining dependencies ($N(\theta) \neq \emptyset$) and non-terminal augmented VNFs without any children. The second type are augmented VNFs with missing parents. When an invalid node is removed from Θ, then the validity of its former parents (Line 30) and children (Line 32) must be verified as well. Forth, the bandwidth and processing requirements of the augmented VNFs are calculated (Line 36). Procedure BCALC uses DP: if the value has been calculated before, then it is retrieved from map b (Line 41), otherwise its value is calculated as the sum of the ingress bandwidths along each of its ingress links (Line 46). Since the ingress bandwidth for an ingress VL $l_{in} \in L_{in}$ is independent of the selected parent during the composition, the bandwidth of the first parent in the AG is used (Line 47). For the use-case, the resulting AG is shown in Figure 6.4.

6.3.3 1-Stage Integer Linear Program (ILP1S)

In this section, we provide a formal description of our problem as an ILP [110], which can be used to find an optimal solution to the combined problem of service composition and

Algorithm 6.1 Procedure to generate the AG.

```
1: var Θ ← ∅, Ψ ← ∅, b ← ∅
2: procedure GENERATE(V, L)
3:     q_1 ← ∅, q_2 ← ∅                                          ▷ 1st phase: initial VNFs
4:     for each v_init ∈ V_init do
5:         θ_new ← new Θ(v=v_init, P=∅, N=∅)
6:         Θ ← [Θ, θ_new]
7:         q_1 ← [q_1, θ_new]
8:         b(θ_new) ← r_init(v)
9:     end for
10:    while |q_1| > 0 do                                        ▷ 2nd phase: other VNFs
11:        θ_cur ← q_1.pop()
12:        v_cur ← v(θ_cur)
13:        for each l_out ∈ L_out^{v_cur} do
14:            for each l_in ∈ L_in^{v_new} : v_new ∈ V, 𝒞_{l_in,l_out} = 1 do
15:                P_new ← P(θ_cur) ∪ l_out ∩ L^next
16:                if L_{l_out}^next ⊂ P_new then
17:                    θ_new ← new Θ(v=v_new, P=P_new, N={θ_cur, L_{l_out}^next})
18:                    Θ ← θ_new ∪ Θ, Ψ ← (θ_cur, θ_new) ∪ Ψ
19:                    q_1 ← q_1 ∪ (θ_new)
20:                    q_2 ← q_2 ∪ (θ_new)
21:                end if
22:            end for
23:        end for
24:    end while
25:    while |q_2| > 0 do
26:        θ_cur ← q_2.pop()                                     ▷ 3rd phase: prune AG
27:        if INVALID(θ_cur) then
28:            Θ ← Θ \ θ_cur
29:            Ψ ← Ψ \ {(θ_s, θ_cur) ∈ Ψ}
30:            q_2 ← q_2 ∪ {θ_s : (θ_s, θ_cur) ∈ Ψ}
31:            Ψ ← Ψ \ {(θ_cur, θ_t) ∈ Ψ}
32:            q_2 ← q_2 ∪ {θ_t : (θ_cur, θ_t) ∈ Ψ}
33:        end if
34:    end while
35:    for θ ∈ Θ do                                              ▷ 4rd phase: bandwidth calculation
36:        BCALC(θ)
37:    end for
38: end procedure
39: procedure BCALC(θ, b)
40:    if θ ∈ KEYS(b) then
41:        return b(θ)
42:    else
43:        b(θ) = 0
44:        for l_in ∈ L_in^{v(θ)} do
45:            for (θ_s, θ) ∈ Ψ^in(θ, l_in) do
46:                b(θ) = b(θ) + r_rel(l_out × (θ_s, θ))BCALC(θ_s)
47:                break
48:            end for
49:        end for
50:        return b(θ)
51:    end if
52: end procedure
```

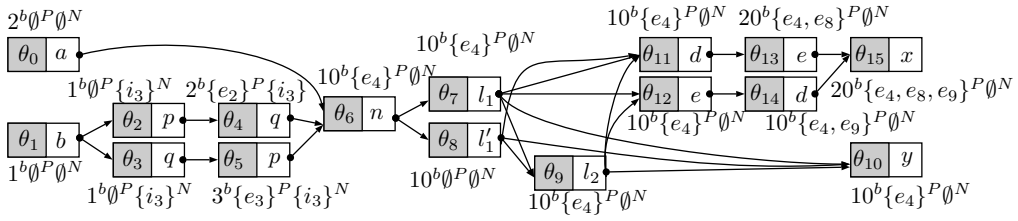


Figure 6.4: AG for the use-case. Empty values of P and N are not shown.

Symbol	Description
Θ	Set of augmented VNFs in the AG.
Ψ	Set of augmented VLs in the AG.
$\nu(\theta) : \Theta \rightarrow V$	VNF corresponding to $\theta \in \Theta$.
$\Psi^{out}(\theta, l_{out}) : \Theta \times L_{out} \rightarrow \Psi$	Set of egress augmented VLs of $\theta \in \Theta$, corresponding to $l_{out} \in L_{out}^{\nu(\theta_1)}$.
$\Psi^{in}(\theta, l_{in}) : \Theta \times L_{in} \rightarrow \Psi$	Set of ingress augmented VLs of $\theta \in \Theta$, corresponding to $l_{in} \in L_{in}^{\nu(\theta)}$.
$b(\psi) : \Psi \rightarrow \mathbb{R}^+$	Bandwidth requirement of augmented VL $\psi \in \Psi$
$d(\theta) : \Theta \rightarrow \mathbb{R}^+$	The processing requirement of augmented VNF $\theta \in \Theta$.

Table 6.4: Input parameters to the service composition for the ILP.

embedding for services with bidirectional chaining requirements and optional VLs. Table 6.4 lists the input parameters related to the VNF chains and service composition. These parameters result from the AG generation algorithm described in Section 6.3.2. The input parameters related to the service embedding are the same as for the original problem (Table 6.3). The decision variables are listed in Table 6.5.

Symbol	Description
X_θ	Integer indicating how many instances of augmented VNF $\theta \in \Theta$ are in the VNF-FG.
Y_ψ	Integer indicating how many instances of augmented VL $\psi \in \Psi$ are in the VNF-FG.
$x_{\theta,n}$	Integer indicating how many instances of augmented VNF $\theta \in \Theta$ are embedded onto PM n.
$y_{\psi,(h_1,h_2)}$	Integer indicating the number of VL instances corresponding to augmented link $\psi \in \Psi$ that are routed along $(h_1,h_2) \in E$.
$\sigma_{\psi,h}$	Integer indicating how many instances of augmented VL $\psi \in \Psi$ originate at $h \in H$.
$\tau_{\psi,h}$	Integer indicating how many instances of augmented VL $\psi \in \Psi$ terminate at $h \in H$.

Table 6.5: Decision variables of the ILP.

Constraints The augmented VNFs corresponding to the terminal VNFs are selected.

$$X_\theta = 1 : \forall \theta \in \Theta | v(\theta) \in V_{init} \cup V_{term} \tag{6.1}$$

The number of embedded VNF instances equals the number of VNF instances in the VNF-FG.

$$X_\theta = \sum_{h \in H} x_{\theta,h} : \forall \theta \in \Theta \tag{6.2}$$

For each augmented VNF instance on a node, the corresponding ingress augmented VL instances are provisioned.

$$\sum_{\psi_{in} \in \Psi^{in}(\theta,l_{in})} \tau_{\psi_{in},h} = x_{\theta,h} : \forall h \in H, \theta \in \Theta, l_{in} \in L_{in}^{v(\theta)} \tag{6.3}$$

For each egress VL $l_{out} \in L_{out}^{v}$ of an instance of VNF $v \in V$, the maximum number of corresponding egress VL instances is $M(l_{out})$. $\forall \theta \in \Theta, l_{out} \in L_{out}^{v(\theta)}, h \in H$:

$$\sum_{\psi_{out} \in \Psi^{out}(\theta,l_{out})} \sigma_{\psi_{out},h} \leq M(l_{out}) x_{\theta,h} \tag{6.4}$$

The processing requirements for the VNFs are assumed additive. If an augmented VNF θ is embedded onto $h \in H$, then its processing requirements $d(\theta)$ are allocated. For each PM $h \in H$, the total processing requirements cannot exceed the node's remaining capabilities.

$$\sum_{\theta \in \Theta} d(\theta) x_{\theta,h} \leq D(h) : \forall h \in H \tag{6.5}$$

VNF $v \in V$ cannot be instantiated on PMs that are not in its candidate set $\phi(v) \subset H$.

$$x_{\theta,h} = 0 : \forall \theta \in \Theta, h \in H \setminus \phi(v(\theta)) \tag{6.6}$$

All VL instances in the VNF-FG are routed through the SNe. Multi-commodity flow constraints dictate that for any augmented VL $(\theta_1, \theta_2) \in \Psi$ the net number of VL instances leaving a certain PM h_1 equals the difference between the number of instances of θ_1 and θ_2 hosted on h_1.

$\forall h_1 \in H, (\theta_1, \theta_2) \in \Psi$:

$$\sum_{(h_1,h_2)\in E} y_{(\theta_1,\theta_2),(h_1,h_2)} - \sum_{(h_2,h_1)\in E} y_{(\theta_1,\theta_2),(h_2,h_1)} = \sigma_{(\theta_1,\theta_2),h_1} - \tau_{(\theta_1,\theta_2),h_1}. \tag{6.7}$$

If augmented VL $\psi \in \Psi$ is routed along PL $(h_1, h_2) \in E$, then its required bandwidth $b(\psi)$ is allocated. The total bandwidth consumption on a PL cannot exceed the available bandwidth capability $B(h_1, h_2)$.

$$\sum_{\psi\in\Psi} b(\psi) y_{\psi,(h_1,h_2)} \leq B(h_1, h_2) : \forall (h_1, h_2) \in E \tag{6.8}$$

Objective function The objective of the resource allocation is to minimize resources consumption, weighted by the scarcity of said resource.

$$\mathbb{L} = \sum_{\theta\in\Theta}\sum_{h\in H} \frac{d(\theta)x_{\theta,h}}{D(h)} + \sum_{\psi\in\Psi}\sum_{(h_1,h_2)\in E} \frac{b(\psi)y_{\psi,(h_1,h_2)}}{B(h_1,h_2)} \tag{6.9}$$

The objective is to minimize $\mathbb{L}$ subject to Equations 6.1 - 6.8.

6.3.4 2-Stage Integer Linear Program (ILP2S)

In this formulation, the optimization is performed in two stages. First, the VNF-FG is composed based on the service requirements, without any knowledge about the SNe capabilities. During composition the number of instances of each augmented VNF and augmented VL is determined. Where in the SNe, these virtualized resources are embedded is not yet considered. In a second stage, this VNF-FG is embedded onto the SNe, taking into account the remaining capabilities. This algorithm is an improvement on related work that performs composition and embedding in two separate stages. For instance, Ocampo et al. solely consider the composition problem for VNF-FGs with a tree topology and without optional VLs. The authors try to find the minimal bandwidth VNF-FG [66]. The authors in [53], [67], [79], [64] assume a precomposed VNF-FG and solely focus on the VNE. The authors in [86] propose a 2-stage approach to the combined problem. In their first stage, a greedy algorithm composes the minimal bandwidth VNF-FG. In their second stage, the VNF-FG is embedded using a MIQCP.

Composition The terminal and initial VNFs are instantiated exactly once (Equation 6.1). Further, each augmented VNF instance requires exactly one ingress VL instances corresponding to each of its ingress VLs in $L_{in}^{v(\theta)}$.

$$\sum_{\psi_{in}\in\Psi^{in}(\theta,l_{in})} I_{\psi_{in}} = X_\theta : \theta \in \Theta, l_{in} \in L_{in}^{v(\theta)}, \tag{6.10}$$

where I_ψ is the number of instances of augmented VL $\psi \in \Psi$. For each instance of $v \in V$, the number of egress VL instances for $l_{out} \in L^v_{out}$ is at most $M(l_{out})$. $\forall \theta \in \Theta, l_{out} \in L^{v(\theta)}_{out}$:

$$\sum_{\psi_{out} \in \Psi^{out}(\theta, l_{out})} I_{\psi_{out}} \leq M(l_{out}) X_\theta \tag{6.11}$$

ILP2S(B) minimizes the VNF-FG VL bandwidth in this stage. The same objective is used in [86, 66].

$$\mathbb{B} = \sum_{\psi \in \Psi} b(\psi) I_\psi \tag{6.12}$$

ILP2S(C) minimizes the processing requirements.

$$\mathbb{C} = \sum_{\theta \in \Theta} d(\theta) X_\theta \tag{6.13}$$

Embedding The resulting values of I_ψ, X_θ from the first stage are added as constraints. Further, for each augmented VL its number of instances equals the times that the augmented VL originates and terminates on any PM in H.

$$I_\psi = \sum_{h \in H} \sigma_{\psi,h} : \forall \psi \in \Psi \tag{6.14}$$

and

$$I_\psi = \sum_{h \in H} \tau_{\psi,h} : \forall \psi \in \Psi \tag{6.15}$$

Finally, the embedding is subject to the constraints in ILP1S and minimizes $\mathbb{L}$ (Equation 6.9).

6.3.5 1-Stage Integer Linear Program (ILP1S-DC)

In this section, we provide a formal description of our problem as an ILP [110], which can be used to find an optimal solution to the combined problem of service composition and embedding for services with bidirectional chaining requirements and optional VLs. The main difference compared to the model presented in Section 6.3.3 is that this model supports a maximum delay for each VL instance corresponding to $\psi \in \Psi$. Further, the model supports for each VNF in $v \in V$, a maximum delay $t^N(v)$ from any initial VNF towards $v \in V$. Typically, these delays are only relevant for terminal VNFs. The calculation of an exact end-to-end delay for each VNF instance quickly becomes unmanageable if more than one instance of an augmented VNF is allowed on a single PM. In this case, one needs to differentiate between all VNF instances on the same PM and decide to which specific VNF instance on another PM these are connected. However, in practical scenarios the collocation of multiple instances of the same augmented VNF on one PM are often required, especially in small SNes. In order, to make the ILP formulation relevant for both small and large problem instances, we will group instances of the same augmented VNF that are collocated on the same PM in delay-groups. Delay-groups correspond to a specific augmented VNF and PM. Instead of calculating the exact delay for each VNF instance, separately, we calculate the upper-bound on the delay of any instance in this delay-group. When the maximum number of VNF instances in the delay-group

Symbol	Description
Θ	Set of augmented VNFs in the AG.
Ψ	Set of augmented VLs in the AG.
$v(\theta) : \Theta \rightarrow V$	VNF corresponding to $\theta \in \Theta$.
$\Psi^{out}(\theta, l_{out}) : \Theta \times L_{out} \rightarrow \Psi$	Set of egress augmented VLs of $\theta \in \Theta$, corresponding to $l_{out} \in L_{out}^{v(\theta_s)}$.
$\Psi^{in}(\theta, l_{in}) : \Theta \times L_{in} \rightarrow \Psi$	Set of ingress augmented VLs of $\theta \in \Theta$, corresponding to $l_{in} \in L_{in}^{v(\theta)}$.
$b(\psi) : \Psi \rightarrow \mathbb{R}^+$	Bandwidth requirement of augmented VL $\psi \in \Psi$
$d(\theta) : \Theta \rightarrow \mathbb{R}^+$	The processing requirement of augmented VNF $\theta \in \Theta$.
$t^L(\psi) : \Psi \rightarrow \mathbb{R}^+$	Maximum delay for augmented VL $\psi \in \Psi$.
$t^N(v) : V \rightarrow \mathbb{R}^+$	Maximum delay from any initial VNF towards $v \in V$.
G	Set of delay-groups.
$\mathcal{G}(\theta) : \Theta \rightarrow G$	Set of possible delay-groups for augmented VNF $\theta \in \Theta$
$h(g) : G \rightarrow H$	PM corresponding to delay-group $g \in G$.
$\mathcal{I}(g)$	Upper bound on the number of VNF instances in delay-group $g \in \mathcal{G}(\theta, h)$, for $\theta \in \Theta$ and $h \in H$
K	Set of ShP indices between any two PMs, i.e. $\{0, 1, \ldots, \lvert K \rvert - 1\}$.
$P_{h_s,h_t,k}$	Ordered set of PLs on the k^{th}-ShP between source PM $h_s \in H$ and target PM $h_t \in H$, for $k \in K$.

Table 6.6: Input parameters to the service composition for the ILP.

equals 1, then the delays calculated in the ILP are exact. Further, the ILP is described in terms of the paths in the SNe between any two PMs, instead of the PLs, to improve scalability. The $|K|$ ShPs between any two PMs in the SNe are generated using the shortest algorithm proposed by Martins et al. [113].

Table 6.6 lists the input parameters related to the VNF chains and service composition. These parameters result from the AG generation algorithm described in Section 6.3.2. The input parameters related to the service embedding are the same as for the original problem (Table 6.3). The decision variables are listed in Table 6.7.

Composition constraints There must be exactly one VNF instance corresponding to each initial and terminal VNF in the VNF-FG.

$$X_\theta = 1 : \forall \theta \in \Theta | v(\theta) \in V_{init} \cup V_{term} \tag{6.16}$$

Each augmented VNF instance requires exactly one ingress VL instance corresponding to each of its ingress VLs in $L_{in}^{v(\theta_t)}$.

$$\sum_{(\theta_s,\theta_t) \in \Psi^{in}(\theta_t, l_{in})} Y_{(\theta_s,\theta_t)} = X_{\theta_t} : \theta_t \in \Theta, l_{in} \in L_{in}^{v(\theta_t)}, \tag{6.17}$$

where Y_ψ is the number of instances of augmented VL $\psi \in \Psi$.

Symbol	Description
X_θ	Integer indicating how many instances of augmented VNF $\theta \in \Theta$ are in the VNF-FG.
Y_ψ	Integer indicating how many instances of augmented VL $\psi \in \Psi$ are in the VNF-FG.
x_g	Integer indicating the number of VNF instances of $\theta \in \Theta$ used in delay-group $g \in \mathscr{G}(\theta)$.
$u_{g_s,g_t,k}$	Integer indicating the number of VL instances between source delay-group $g_s \in \mathscr{G}(\theta_s)$ and target delay-group $g_t \in \mathscr{G}(\theta_t)$ that are routed along the k^{th} ShP between $h(g_s)$ and $h(g_t)$, where $k \in K$.
$U_{g_s,g_t,k}$	Binary indicating if VL instances between source delay-group $g_s \in \mathscr{G}(\theta_s)$ and target delay-group $g_t \in \mathscr{G}(\theta_t)$ are routed along the k^{th} ShP between $h(g_s)$ and $h(g_t)$, where $k \in K$.
C_{g_s,g_t}	Binary indicating if there are VL instances between source delay-group $g_s \in \mathscr{G}(\theta_s)$ and target delay-group $g_t \in \mathscr{G}(\theta_t)$, for $(\theta_s,\theta_t) \in \Psi$.
Δ_g	Double indicating the maximum delay from any initial VNF towards delay-group $g \in \mathscr{G}(\theta), \theta \in \Theta$.
δ_{g_s,g_t}	Double indicating the product $\Delta_{g_s} C_{g_s,g_t}$ for $g_s \in \mathscr{G}(\theta_s), g_t \in \mathscr{G}(\theta_t), (\theta_s,\theta_t) \in \Psi$.

Table 6.7: Decision variables of the ILP.

For each instance of $v \in V$, the number of egress VL instances for $l_{out} \in L^v_{out}$ is at most $M(l_{out})$. $\forall \theta_s \in \Theta, l_{out} \in L^{v(\theta_s)}_{out}$:

$$\sum_{(\theta_s,\theta_t) \in \Psi^{out}(\theta_s,l_{out})} Y_{(\theta_s,\theta_t)} \leq M(l_{out}) X_{\theta_s} \tag{6.18}$$

Embedding constraints The number of embedded VNF instances equals the number of VNF instances in the VNF-FG.

$$X_\theta = \sum_{g \in \mathscr{G}(\theta)} x_g : \forall \theta \in \Theta \tag{6.19}$$

The number of VNF instances for each delay-group g cannot exceed $\mathscr{I}(g)$.

$$x_g \leq \mathscr{I}(g) : \forall g \in \mathscr{G}(\theta), \theta \in \Theta \tag{6.20}$$

The k^{th} ShP between $h_s = h(g_s)$ and $h_t = h(g_t)$, i.e. $P_{h_s,h_t,k}$, is used for communication between source delay-group g_s and target delay-group g_t if at least one of the VL instances between these groups uses this path. $\forall (\theta_s,\theta_t) \in \Psi, g_s \in \mathscr{G}(\theta_s), g_t \in \mathscr{G}(\theta_t), k \in K$:

$$u_{g_s,g_t,k} \leq \mathscr{I}(g_t) U_{g_s,g_t,k} \tag{6.21}$$

There is communication between source delay-group g_s and target delay-group g_t if at least one VL instance interconnects g_s and g_t over any ShP. $\forall (\theta_s,\theta_t) \in \Psi, g_s \in \mathscr{G}(\theta_s), g_t \in \mathscr{G}(\theta_t)$:

$$\sum_{k \in K} u_{g_s,g_t,k} \leq |K| C_{g_s,g_t} \tag{6.22}$$

Further on, auxiliary decision variables $U_{g_s,g_t,k}$ and C_{g_s,g_t} will be used to calculate the delay of each delay-group. For each VNF instance in a delay-group, the corresponding ingress VL instances are provisioned. $\forall \theta_t \in \Theta, g_t \in \mathcal{G}(\theta_t), l_{in} \in L_{in}^{v(\theta_t)}$:

$$\sum_{k \in K} \sum_{(\theta_s,\theta_t) \in \Psi^{in}(\theta_t,l_{in})} \sum_{g_s \in \mathcal{G}(\theta_s)} u_{g_s,g_t,k} = x_{g_t} \tag{6.23}$$

The delay for a delay-group corresponding to an initial VNF equals this initial VNF's processing delay. $\forall \theta_{ini} \in \Theta | v(\theta) \in V_{ini}, g \in \mathcal{G}(\theta_{ini})$:

$$\Delta_g = T^N(v(\theta_{ini}), b(\theta_{ini})) \tag{6.24}$$

Further, the worst-case delay from any initial VNF towards any VNF instance in delay-group g_t is the maximum of the delay of any source delay-group g_s that it is connected to, plus the highest propagation delay from this source-group towards g_t and the processing delay for the target VNF instance. $\forall (\theta_s, \theta_t) \in \Psi, g_s \in \mathcal{G}(\theta_s), g_t \in \mathcal{G}(\theta_t), h_s = h(g_s), h_t = h(g_t), k \in K$:

$$\Delta_{g_t} \geq \delta_{g_s,g_t} + \sum_{e \in P_{h_s,h_t,k}} T^L(e) U_{g_s,g_t,k} + T^N(v(\theta_t), b(\theta_t)), \tag{6.25}$$

where $\forall (\theta_s, \theta_t) \in \Psi, g_s \in \mathcal{G}(\theta_s), g_t \in \mathcal{G}(\theta_t)$:

$$\delta_{g_s,g_t} = \Delta_{g_s} C_{g_s,g_t}. \tag{6.26}$$

This product indicates the delay contribution of delay-group g_s on g_t; it can be linearized by adding the following equations. $\forall (\theta_s, \theta_t) \in \Psi, g_s \in \mathcal{G}(\theta_s), g_t \in \mathcal{G}(\theta_t)$:

$$\delta_{g_s,g_t} \leq \mathcal{T} C_{g_s,g_t}, \tag{6.27}$$

where $\mathcal{T} = \max_{v \in V} t^N(v)$;

$$\delta_{g_s,g_t} \leq \Delta_{g_s}; \text{ and} \tag{6.28}$$

$$\delta_{g_s,g_t} \geq \Delta_{g_s} - (1 - C_{g_s,g_t})\mathcal{T}. \tag{6.29}$$

For each delay-group, the total delay cannot exceed the maximum allowed delay of the corresponding VNF.

$$\Delta_g \leq t^N(v(\theta)) : \forall \theta \in \Theta, g \in \mathcal{G}(\theta) \tag{6.30}$$

For each egress VL $l_{out} \in L_{out}^v$ of an instance of $v \in V$, the maximum number of corresponding egress VL instances is $M(l_{out})$. $\forall \theta_s \in \Theta, g_s \in \mathcal{G}(\theta_s), l_{out} \in L_{out}^{v(\theta_s)}$:

$$\sum_{k \in K} \sum_{(\theta_s,\theta_t) \in \Psi^{out}(\theta_s,l_{out})} \sum_{g_t \in \mathcal{G}(\theta_t)} u_{g_s,g_t,k} \leq M(l_{out}) x_{g_s} \tag{6.31}$$

The processing requirements for the VNFs are assumed additive. If an augmented VNF θ is embedded onto $h \in H$, then its processing requirements $d(\theta)$ are allocated. For each PM $h \in H$, the total processing requirements cannot exceed the node's remaining capabilities.

$$\sum_{\theta \in \Theta} \sum_{g \in \mathcal{G}(\theta) : h(g) = h} d(\theta) x_g \leq D(h) : \forall h \in H \tag{6.32}$$

If augmented VL $\psi \in \Psi$ is routed along PL $(h_1, h_2) \in E$, then its required bandwidth $b(\psi)$ is allocated. The total bandwidth consumption on a PL cannot exceed the available bandwidth capability $B(e)$. $\forall e \in E, h_s = h(g_s), h_t = h(g_t), K^* = \{k \in K : e \in P_{h_s,h_t,k}\}$:

$$\sum_{(\theta_s,\theta_t)\in\Psi} \sum_{g_s\in\mathcal{G}(\theta_s)} \sum_{g_t\in\mathcal{G}(\theta_t)} \sum_{k\in K^*} b(\psi) u_{g_s,g_t,k} \leq B(e) \tag{6.33}$$

An augmented VL cannot be routed over a path in the SNe, which introduces a delay that exceeds the maximum allowed delay for this VL instance. $\forall(\theta_s, \theta_t) \in \Psi, g_s \in \mathcal{G}(\theta_s), g_t \in \mathcal{G}(\theta_t), k \in K, \sum_{e\in P_{h_s,h_t,k}} T^L(e) > t^L(\theta_s, \theta_t)$:

$$u_{g_s,g_t,k} = 0 \tag{6.34}$$

The number of instances of each augmented VL in the VNF-FG is the sum of all VL instances between any two delay-groups.

$$Y_{(\theta_s,\theta_t)} = \sum_{g_s\in\mathcal{G}(\theta_s)} \sum_{g_t\in\mathcal{G}(\theta_t)} \sum_{k\in K} u_{g_s,g_t,k} : \forall(\theta_s, \theta_t) \in \Psi \tag{6.35}$$

Objective function The objective of the resource allocation is to minimize resources consumption, weighted by the scarcity of said resource.

$$\begin{aligned} \mathbb{L} = & \sum_{\theta\in\Theta} \sum_{g\in\mathcal{G}(\theta)} \frac{d(\theta)x_g}{D(h(g))} \\ + & \sum_{(\theta_s,\theta_t)\in\Psi} \sum_{g_s\in\mathcal{G}(\theta_s)} \sum_{g_t\in\mathcal{G}(\theta_t)} \sum_{k\in K: e\in P_{h_s,h_t,k}} \frac{b(\theta_s,\theta_t)u_{g_s,g_t,k}}{B(e)} \end{aligned} \tag{6.36}$$

The objective is to minimize $\mathbb{L}$ subject to Equation 6.16 - 6.33.

6.3.6 2-Stage Integer Linear Program (ILP2S-DC(f))

In this formulation, the optimization is performed in two stages. The main difference compared to the model presented in Section 6.3.4 is that this model supports a maximum delay for each VL instance corresponding to $\psi \in \Psi$. Further, the model supports for each VNF in $v \in V$, a maximum delay $t^N(v)$ from any initial VNF towards $v \in V$.

Composition ILP2-DC(B) minimizes the VNF-FG VL bandwidth subject to Equation 6.16 through 6.18 in this stage. The same objective is used in [86] and [66].

$$\mathbb{B} = \sum_{\psi\in\Psi} b(\psi) Y_\psi \tag{6.37}$$

ILP2-DC(C) minimizes the processing requirements.

$$\mathbb{C} = \sum_{\theta\in\Theta} d(\theta) X_\theta \tag{6.38}$$

Algorithm 6.2 Recursive heuristic.

```
1:  var G(θ,Ψ), G(H,E), B, D, κ, a
2:  a ← 0
3:  M ← []
4:  procedure REC(M)
5:      if FINISHED(M) then
6:          return M                                   ▷ Success
7:      else if a > α then
8:          return ∅                                   ▷ Backtrack limit exceeded
9:      end if
10:     C ← GENCANDIDATES(M)                           ▷ Generate candidates
11:     for each c ∈ C do
12:         M' ← [M, c]                                ▷ Add c to the mapping
13:         M'' ← REC(M')                              ▷ Recurse
14:         if M'' ≠ ∅ then
15:             return M''                             ▷ Sucessful recursion
16:         end if
17:         a ← a + 1                                  ▷ Backtrack
18:     end for
19:     return ∅                                       ▷ Failure
20: end procedure
```

Embedding The resulting values of Y_ψ, X_θ from the first stage are added as constraints. Finally, the embedding is subject to Equation 6.19 through 6.35 and minimizes $\mathbb{L}$ (Equation 6.36).

6.4 Recursive heuristic (REC)

Given the computational intractability of the exact algorithms presented in the previous section, a recursive heuristic for the combined VNF-FG composition and embedding problem is proposed.

6.4.1 Algorithm description

REC (Algorithm 6.2) is a recursive procedure that terminates as soon as a valid mapping is found. REC makes use of the GENCANDIDATES (Algorithm 6.3) function, generating the candidates that can be added to the current mapping and the FINISHED function, inquiring whether the current mapping satisfies all SRs.

The algorithm is an improvement on the recursive heuristic by Beck et al. [85], which does not consider optional VLs, traffic aggregation, bidirectional chaining requirements and delay constraints. This heuristic decides at the same time, which VL instance to add to the VNF-FG and how to embed it. In order to avoid exploring all possible PCs, which would be computationally intractable, the search space has to be limited. Because low-cost PCs tend to avoid routing VLs over very long paths in the SNe, it makes sense to only consider a subset of the possible candidates at each step of the algorithm.

Parameter κ limits the size of the candidate set to consider in each recursion step. Parameter α limits the maximum number of backtracks that the algorithm can perform. Upon the algorithm's initiation, the number of backtracks that have been performed (a), is set to 0

Algorithm 6.3 Candidates generating procedure.

```
1:  var G(θ,Ψ), G(H,E), B, D, κ
2:  procedure GENCANDIDATES(M)
3:      (θ_inst, l_in) =GETMISSINGINGRESSVL(M)
4:      if θ_inst != ∅ then                          ▷ Add ingress VL instance
5:          return INGRESSBFS(M, θ_inst, l_in)
6:      else                                         ▷ Add terminal VNF instance
7:          θ_term =GETMISSINGTERMVNF(M)
8:          C ← []
9:          for h ∈ Ψ(v(θ_term)) do
10:             if D(h) ≥ d(θ_term) then
11:                 C ← [C, ((θ_term, θ_term), [h → h])]
12:             end if
13:         end for
14:         return C
15:     end if
16: end procedure
```

(Line 2) and the mapping ($\mathcal{M}$) is empty (Line 3). Variable $\mathcal{M}$ represent the current mapping as a list of augmented VL to PP mappings.

REC($\mathcal{M}$) tries to find a valid mapping by extending $\mathcal{M}$. If REC($\mathcal{M}$) finds a valid mapping, then it returns this mapping (Line 6). An empty set is returned if the backtracking limit has been exceeded (Line 8), or if a valid mapping cannot be found by extending the current mapping ($\mathcal{M}$) (Line 19).

If the current mapping ($\mathcal{M}$) does not satisfy all SRs and the backtrack limit has not been exceeded, then the algorithm iterates over all possible candidates $c \in C$. In each iteration, c is added to $\mathcal{M}'$, i.e. a copy of $\mathcal{M}$ (Line 12). Next, the algorithm recursively searches for a valid mapping by extending $\mathcal{M}'$ (Line 13). If the resulting mapping ($\mathcal{M}''$) is valid, then this mapping is returned. Otherwise, the number of backtracks is incremented and the next candidate in C is tried. If all candidates have been exhausted, then the algorithm returns $\emptyset$.

The GENCANDIDATES($\mathcal{M}$) function generates the (VL, PP) mappings that can be added to $\mathcal{M}$. This procedure is described in Algorithm 6.3.

Initially, GETMISSINGINGRESSVL($\mathcal{M}$) searches for the first VNF instance that is missing an ingress VL in $\mathcal{M}$, together with this missing VL. GETMISSINGINGRESSVL($\mathcal{M}$) traverses the VNF instances in $\mathcal{M}$ in order that they were added, and returns the first VNF instance $\theta_{inst} \in \Theta(\mathcal{M})$ with missing parents via $l_{in} \in L_{in}^{v(\theta_{inst})}$ that it encounters. If no missing combination is found, $(\emptyset, \emptyset)$ is returned. Based on the presence of an augmented VNF instance with a missing ingress VL instance, one can distinguish between two cases. On the one hand, if an ingress VL is missing, the candidate VL instances to resolve this missing parent together with their possible PP embeddings are generated by INGRESSBFS($\mathcal{M}, \theta_{inst}, l_{in}$). This function performs a BFS search combined with a look-ahead mechanism that verifies Equations 6.16, 6.17 and 6.18, to filter the candidates based on the current number of instances of each augmented VNF and VL in $\mathcal{M}$. This iterative procedure keeps track at each step of the upper bounds on the number of instances of each augmented VL and VNF instance and the total maximum of terminal VNF instances and calculates the corresponding values for each candidate augmented link. When the maximum instances of a particular VNF is reduced to 0, or the total maximum of terminal VNF instances is smaller than the cardinality of $|V_{term}|$, then an augmented link is no

valid candidate. These upper bounds are calculated as follows. When $\mathscr{M}$ is empty, the upper bounds are all set to ∞. Then, the upper bound on the number of instances for the initial VNFs are set to 1 (Equation 6.16). For instance, in the AG shown in Figure 6.4, $X_{\theta_0} \leq 1$ and $X_{\theta_1} \leq 1$. Subsequently, each time an upper bound changes (decreases), the upper-bounds for its children and siblings in the AG are re-evaluated (Equations 6.17 and 6.18). Since $\theta_0 \leq 1$ and $M(e_0) = 1$, it follows that $Y_{(\theta_0,\theta_1)} \leq 1$. Since each ingress VL of n must correspond to exactly one VL instance and θ_6 has only a single parent via i_2, $X_{\theta_6} \leq 1$. From $M(e_4) = 1$, it follows that $X_{\theta_7} \leq 1$ and $X_{\theta_8} \leq 1$, and so on. Since, θ_7 and θ_8 are siblings, they compete for egress VL e_4 of θ_6. When in a later instance, either (θ_6, θ_7), or (θ_6, θ_8) is added to the VNF-FG, it follows that the maximum number of instances of the other one is reduced to 0. Verifying the upper bound on the total number of possible target VNFs is important, to avoid unnecessary backtracks when the caching would not provide a high enough fan-out later on. For instance, when an instance of θ_7 is selected, at least one instance of θ_9 has to be connected to θ_7 to serve more than 5 target VNF instances.

INGRESSBFS generates candidate PPs targeted to the PM on which this VNF instance with missing parents is hosted, for each of these candidate ingress VLs. The BFS procedure finds the closest PMs that can host the source VNF of this VL instance, while considering remaining bandwidth and processing capabilities. The procedure returns at most κ candidates, located at at-most 7 hops from the target VNF instance. Finally, these candidates are sorted in order or increasing cost, as defined by Equation 6.36. On the other hand, when no VNF instance in $\mathscr{M}$ is missing a parent, then the next missing terminal VNF $v_{term} \in V_{term}$ is added. In this case, the candidate PMs in $\psi(v_{term})$ that have sufficient processing capability remaining to host this VNF are added to the candidate list (Line 11).

The FINISHED($\mathscr{M}$) function verifies in the REC procedure (Line 5) if the SRs have been satisfied. If the mapping ($\mathscr{M}$) is not missing any ingress VL or terminal VNF instance, then this mapping satisfies the SRs and the function returns TRUE. Otherwise FINISHED returns FALSE.

6.4.2 Illustration

The execution of the recursive algorithm is illustrated using the use-case; its SRs, target SN and corresponding AG are depicted in Figure 6.1, 6.3 and 6.4 respectively. The steps of the algorithm are listed in Table 6.8. The resulting PC of REC for a maximum end-to-end delay of 270 ms for each terminal VNF is shown in Figure 6.3. Since, the minimal bandwidth VNF-FG cannot be embedded while respecting this delay-limit, the resulting PL for VNF-FG$_1$ (see Figure 6.2a) is shown for a maximum end-to-end delay of 500 ms.

The mapping steps are numbered ($\mathscr{S}$). Candidate c is the VL instance (θ_s, θ_t) to PP mapping that is added to $\mathscr{M}$ in that step and $cost(c)$ denotes its contribution to the objective function (Equation 6.36). The remaining delay budget for the source VNF instance of c is given by $\mathscr{R}(\theta_s)$. The mapping step relative to which c is added is given by $O(\mathscr{S})$. When the difference between $\mathscr{S}$ and $O(\mathscr{S})$ exceeds one, the algorithm has backtracked, indicating that the mappings between $\mathscr{S}$ and $O(\mathscr{S})$ have been removed. In this case $O(\mathscr{S})$ is underlined.

Step 0 corresponds to an empty mapping, i.e., $\mathscr{M} = \emptyset$. In step 1, no ingress VLs are missing and an instance of terminal VNF x is added; its only candidate PM is x_2. The cost

$\mathscr{S}$	c	$cost(c)$	$\mathscr{R}(\theta_s)$	$O(\mathscr{S})$
1	$(\theta_{15},\theta_{15}) \rightarrow [x_2, x_2]$	0.20	270	0
2	$(\theta_{14},\theta_{15}) \rightarrow [d_3, X_3, X_2, x_2]$	0.70	208	1
3	$(\theta_{12},\theta_{14}) \rightarrow [e_2, X_2, X_3, d_3]$	0.60	136	2
4	$(\theta_9,\theta_{12}) \rightarrow [l_3, X_3, X_2, e_2]$	0.70	64	3
5	$(\theta_7,\theta_9) \rightarrow [l_3, l_3]$	0.60	54	4
6	$(\theta_6,\theta_7) \rightarrow [n_3, X_3, l_3]$	0.40	42	5
7	$(\theta_7,\theta_{12}) \rightarrow [l_3, X_3, X_2, e_2]$	0.90	64	$\underline{3}$
8	$(\theta_6,\theta_7) \rightarrow [n_3, X_3, l_3]$	0.40	52	7
9	$(\theta_{13},\theta_{15}) \rightarrow [e_2, X_2, x_2]$	1.00	258	$\underline{1}$
10	$(\theta_{11},\theta_{13}) \rightarrow [d_3, X_3, X_2, e_2]$	0.70	186	9
11	$(\theta_9,\theta_{11}) \rightarrow [l_3, X_3, d_3]$	0.60	164	10
12	$(\theta_7,\theta_9) \rightarrow [l_3, l_3]$	0.60	154	11
13	$(\theta_6,\theta_7) \rightarrow [n_3, X_3, l_3]$	0.40	142	12
14	$(\theta_0,\theta_6) \rightarrow [a_1, X_1, X_3, n_3]$	0.14	60	13
15	$(\theta_4,\theta_6) \rightarrow [q_1, X_1, X_3, n_3]$	0.58	60	14
16	$(\theta_5,\theta_6) \rightarrow [p_3, X_3, n_3]$	1.02	110	$\underline{14}$
17	$(\theta_3,\theta_5) \rightarrow [q_1, X_1, X_3, p_3]$	0.29	48	16
18	$(\theta_1,\theta_3) \rightarrow [b_1, X_1, q_1]$	0.04	26	17
19	$(\theta_{10},\theta_{10}) \rightarrow [y_3, y_3]$	0.10	270	18
20	$(\theta_7,\theta_{10}) \rightarrow [l_3, X_3, y_3]$	0.20	154	19

Table 6.8: Illustration of the execution of REC.

is the ratio of the instance's CPU requirements to the remaining PM capability of x_2, i.e. $20/100 = 0.2$. The remaining budget delay budget is the delay budget of x, i.e. 270 ms. In step 2, it is detected that x_2 misses a connection to i_7. Hence, the candidate source augmented VNFs are θ_{14} and θ_{13}, which can be mapped to either PM d_3 and e_2 respectively. Augmented VNF θ_{14} requires 10 processing units; θ_{15} requires an ingress bandwidth of 20 bandwidth units over 3 hops. Hence, the cost for θ_{14} equals $10/100 + 20/100 \times 3 = 0.7$, which is less than for source augmented VNF θ_{13}. The remaining budget for θ_{14} equals the remaining budget for θ_{15} minus the processing delay of θ_{15} and the routing delay for $(\theta_{14}, \theta_{15})$, i.e. $270\,\text{ms} - 10\,\text{ms} - (1\,\text{ms} + 50\,\text{ms} + 11\,\text{ms}) = 208\,\text{ms}$. In step 3, the ingress VL of θ_{14}, i.e. i_{10} is missing. The only augmented VL candidate for this VL instance is $(\theta_{12}, \theta_{14})$, which can only be hosted on e_2. In step 4, VNF l_2 is placed onto PM l_3 since it is the candidate with the lowest cost. In step 5, VNF l_1 is placed onto PM l_3. In step 6, VNF n is placed onto PM n_3, with a remaining delay budget of 42 ms. Insufficient delay budget remains to add the first missing ingress VL instance of n, i.e., i_2. The remaining budget for an instance of a should at least equal the processing delay of a, which is 20 ms. The processing delay of n is 20 ms and no candidate source PM can be found for θ_0 with the remaining delay budget of 22 ms. Therefore, the algorithm backtracks to the mapping $\mathscr{M}$ after step 3.

In step 7, θ_{12} is directly connected to VNF l_1, instead of via l_2. The embedding cost is higher than in step 4, namely 0.9 versus 0.7 because the generated candidates are considered in order of increasing cost. In step 8, VNF n is again placed on PM n_3. Compared to step 6, the delay budget of n is now slightly increased. The delay budget still does not suffice to connect to an instance of VNF a placed onto PM a_1. Therefore, the algorithm backtracks to the mapping of step 1.

In step 9, VNF e is chained before VNF x. In step 10, VNF d is chained before VNF e, effectively reversing the sequence tried before. In step 11, 12 and 13, VNFs l_2, l_1 and n are mapped again. Note that due to alteration of the relative order of VNF d and e that the delay budget of n is now 142 ms, instead of 42 ms. In step 14, enough delay budget (60 ms) remains to place VNF a on PM a_1. In step 15, the next missing ingress VL of n, i.e. i_3 is addressed. VNF q is chained before VNF n. The remaining budget (60 ms) is however not enough to connect a p instance before q and the algorithm backtracks.

In step 16, VNF p is chained before n, instead op q. In step 17 and 18 VNF q and b can be chained. In step 19, no mapped VNFs are missing ingress VLs and the second and last terminal VNF is mapped to PM y_3. Finally, in step 20, the ingress VL of VNF y, i.e. i_8 is connected to l_1. No mapped VNFs have missing ingress VLs and all terminal VNFs are mapped, FINISHED($\mathscr{M}$) returns true and $\mathscr{M}$ is returned.

6.5 Performance Evaluation

6.5.1 Requirements

Composition requirements The generated SRs are based on the use-case presented in Section 6.3. The number of source and target VNFs are both uniformly distributed in 1 to 10. Source and target VNF have a single egress and ingress VL respectively. Each source VNF is with a probability of 50% directly connected to VNF n, else it is connected through a

chain of pre-processing VNFs to the n. The VNFs in these chains are unique, i.e. a VNF can only appear in a single pre-processing or post-processing chain. All VNFs in these chains are required. The number of VNFs in every pre- and post-processing chain are all independent and uniformly distributed in 1 to 5. The caching VNF configuration and requirements are identical to the use-case. For the other VNFs, $r_{rel}(l)$ and $d_{rel}(v)$ are independent and uniformly selected from $\{0.75, 1.0, 1.5\}$ and $\{1, 2, 3\}$ respectively. The initial bandwidths are all independent and uniformly selected from $\{1, 2, 3\}$.

Embedding constraints Transit-stub SGs are generated using the GT-ITM topology generator on a 100×100 grid [106]. Any two nodes in the transit network are connected by a PL with a probability of 80%. Within a stub-network cluster this probability is 40%. Each PM in the transit network is connected to 2 clusters, each comprising 9 PMs. Unless stated otherwise, the number of transit nodes is 1, corresponding to 19 PMs. The PL bandwidths are uniformly distributed with average B_{avg} and the ratio between the upper and lower range of the interval equals 5. Similarly, the PM capabilities are uniformly distributed with average D_{avg}, equal to $D_{avg} = 100000$. The delay introduced by a PLs $(h_1, h_2) \in E$ is assumed proportional to the Euclidean distance $dist(h_1, h_2)$ between the two end-points. The delay corresponding to the width of the grid is assumed 100 ms. The processing delays are assumed independent of the PM on which they are executed. The processing delays for all VNFs are independent and uniformly selected from $\{10, 20, 30\}$.

The LCs are generated in the following way, a similar approach is taken in [78, 49], [C2]: the VNFs are each assigned a preferred location, which corresponds to the location of a randomly selected PM $h_{pref}(v) \in H$. The sources and targets can only be located on h_{pref}, the other VNFs can with a probability of 80% be located on any of the PMs that are within a Euclidian distance ρ from $h_{pref}(v) \in H : \forall v \in V$.

$$\phi(v) = \{h \in H | X_h \leq 0.8 \cap dist(h, h_{pref}(v)) \leq \rho\}, \tag{6.39}$$

where all $X_h : h \in H$ are independent and uniformly distributed in $[0; 1]$ and $dist(h_1, h_2)$ is the Euclidean distance between PMs h_1 and h_2. The location constraints model a combination of geographical and hardware related limitations. Further, the candidate set $\psi(v)$ for $v \in V$ must at least contain one PM.

Service arrival Request arrival and service-rates are modeled as Poisson processes. The arrival-rate (λ) equals 1 arrival per unit of time. The average duration of a request is the inverse of its service rate (μ). For each parameter configuration, 10 random SNes are generated using GT-ITM, each with a random assignment of location constraints and composition requirements. In the following we distinguish between an offline scenario and an online scenario, in both the requests are processed one-by-one. However, in the offline scenario, 100 requests are processed that are active until the end of the simulation, i.e. $\mu \to 0$, corresponding the scenario where the cloud environment is initiated, and the pending requests are processed. In the online scenario more requests are processed; in steady-state 100 requests are expected to be active at the same time since $\frac{\lambda}{\mu} = 100$.

Algorithm	DCs	Coordinated	Section	Routing
ILP1S	no	yes	6.3.3	IMCF
ILP2S(f)	no	no	6.3.4	IMCF
ILP1S-DC	yes	yes	6.3.5	$\|K\|$-ShP
ILP2S-DC(f)	yes	no	6.3.6	$\|K\|$-ShP
REC	yes	yes	6.2	BFS

Table 6.9: Evaluated algorithms.

6.5.2 Metrics

The following metrics are evaluated. The *acceptance ratio* indicates the fraction of requests that is accepted. For a given service request, the acceptance ratio is 0 or 1 for rejection and acceptance, respectively. The SG *bandwidth* is the total of the bandwidth resources consumed in the SNe to route the VLs, per accepted request. The *processing consumption* is the total of the CPU resources consumed, per accepted request. The CPU consumption of a given VNF-FG is independent of the embedding. The *computation time* is the time required by the algorithm to process a service request.

6.5.3 Evaluated algorithms

The evaluated algorithms are listed in Table 6.9. The ILPs are solved using Gurobi 7.5.2, a hybrid solver, combining multiple solution techniques [114]. The ILP-based algorithms use 4 threads. If no solution can be found within 10 minutes, the request is rejected.

ILP1S, ILP2S(B), ILP2S(C) the algorithms presented in Section 6.3.3 and 6.3.4, using Integer MCF (IMCF) routing; the algorithms do not support delay constraints. ILP1S solves the combined composition and embedding problem in one stage and can thus reach the global objective. ILP2S(f) solves the composition and embedding problems in two separate stages, it minimizes function f in the first stage. Both stages are not coordinated. ILP2S(B) and ILP2S(C) minimize the VNF-FG bandwidth and processing requirements respectively.

ILP1S-DC and ILP2S-DC(f) the algorithms presented in Section 6.3.5 and 6.3.6 respectively, based on path-generation and supporting delay-constraints. Unless specified otherwise, the maximum number of VNF instances per delay group $\mathscr{I} = 1$ for each delay-group $g \in G$. Further, a single ShP is generated between any two PMs in H, i.e. $|K| = 1$.

REC the recursive heuristic that coordinates composition and embedding, presented in Section 6.4. GENCANDIDATES generates at most 10 candidates per iteration, i.e., $\kappa = 10$. Further, the algorithm performs at most 10000 backtracks, i.e., $\alpha = 10000$.

6.5.4 Simulation parameters

The key simulation parameters are:

- ρ: geographical distance (radius) from the preferred PM, where a VNF can be placed. It is a measure for the flexibility in the embedding: a low ρ corresponds to a very location-constrained embedding;

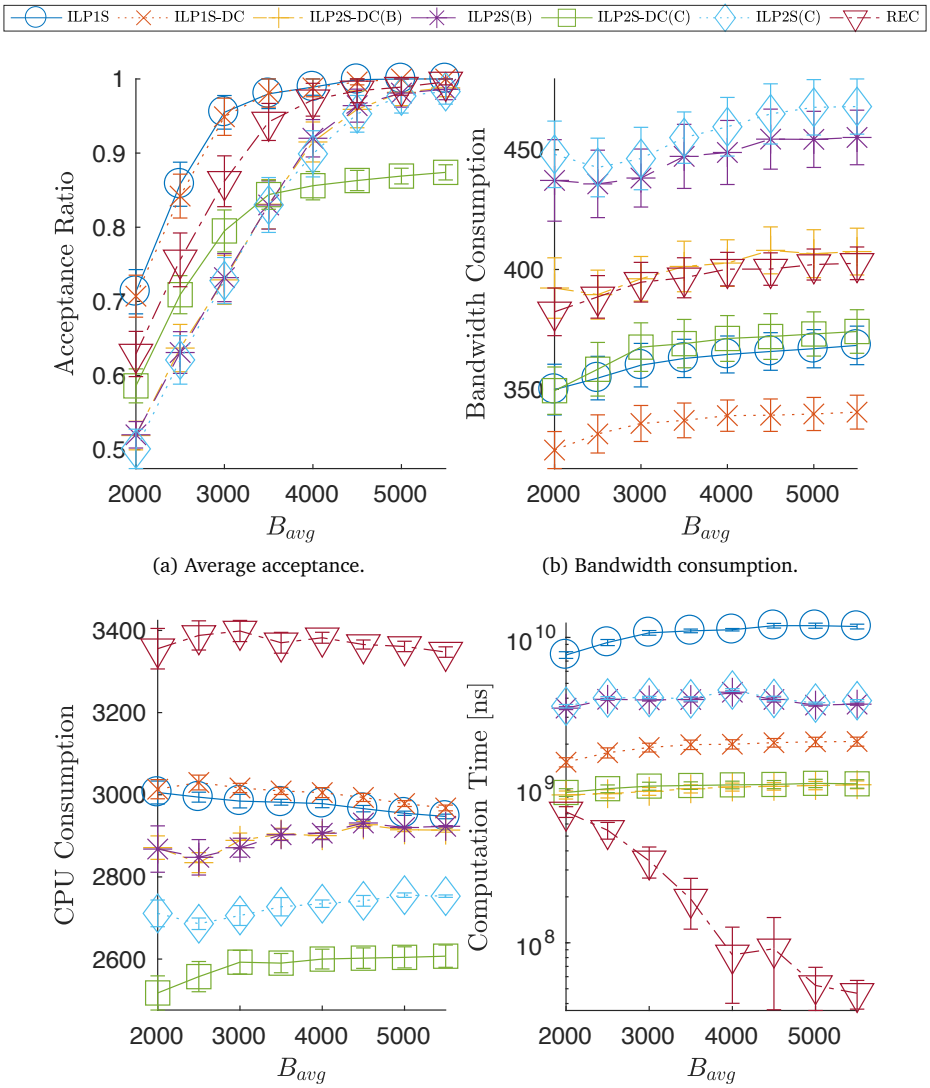
- B_{avg}: the average PL bandwidth capability in the SNe.

- t^N_{max}: the maximum allowed end-to-end delay for the terminal VNFs.

6.5.5 Results

Offline The impact of the average PL bandwidth capability B_{avg} is shown in Figure 6.5. In this setup, there are no delay limitations, i.e. $t^L_{max} \to \infty$, $t^N_{max} \to \infty$ and $\rho = 5$. Overall, the acceptance ratio goes up as B_{avg} increases and it is the highest for ILP1S and ILP1S-DC. The acceptance ratio for the 1-phase ILP is up to 42% higher than for the 2-phase ILPs, while requiring only 11× more time. Even though ILP1S-DC considers only a single ShP between any two PMs, its acceptance ratio is only 2% lower than for ILP1S. Interestingly, the ShP routing for ILP1S-DC results in a SN bandwidth consumption that is up to 8% lower than for ILP1S. The acceptance ratio for REC is up to 12% lower than for ILP1S, but it can find a solution up to 252× faster. As the available PL bandwidth increases, REC performs fewer recursions to find a feasible solution. The acceptance ratio for REC is up to 13% higher than for ILP2S-DC(C) and up to 21% higher than for ILP2S-DC(B). While the performance of ILP2S-DC(B) and ILP2S(B) are very similar, the performance of ILP2S-DC(C) is very different than for ILP2S(C). The reason is that ILP2S-DC(C) can only use the ShP between any two PMs. Since the algorithm minimizes the CPU resources instead of the VL bandwidth, the available bandwidth on some PLs becomes a bottleneck, especially when the LCs are very strict ($\rho = 5$). ILP2S(C) can avoid these overloaded links during routing, while ILP2S-DC(C) cannot. The VNF-FG with minimal CPU requirements always features a caching hierarchy comprising an l_1 instance and possibly multiple l_2 instances that need to be interconnected, while the other algorithms can use the larger l'_1 instance, curbing the performance of ILP2-DC(C) for $|K| = 1$.

Whether a VNF-FG with minimal VL bandwidth or processing requirements is easiest to embed, or which default composition strategy performs best depends strongly on both the application and cloud environment. Hence, the performance of a 2-stage approach with a given objective in the composition stage is dependent on the SNe conditions. Further, REC explores multiple compositions only when it backtracks. Hence, its computation time and placement quality depend on the order in which the candidates in GENCANDIDATES are ordered. As an illustration consider its CPU consumption, which is up to 14% higher than for ILP1S. The reason for this high CPU consumption is that the algorithm often places additional l_2 caches that are not particularly useful, such as in Figure 6.2a. Improvements to GENCANDIDATES, which are considered out of scope for this article, can avoid this wasteful behavior. Overall, the SG bandwidth consumption increases for all algorithms as the available PL bandwidth increases. Interestingly as the available PL bandwidth increases, the CPU consumption per accepted request *decreases* slightly for the coordinated approaches, while it increases for the uncoordinated ones. Since most of the algorithms can place all requests for $B_{avg} = 6000$, this value will be used for the remaining simulations.



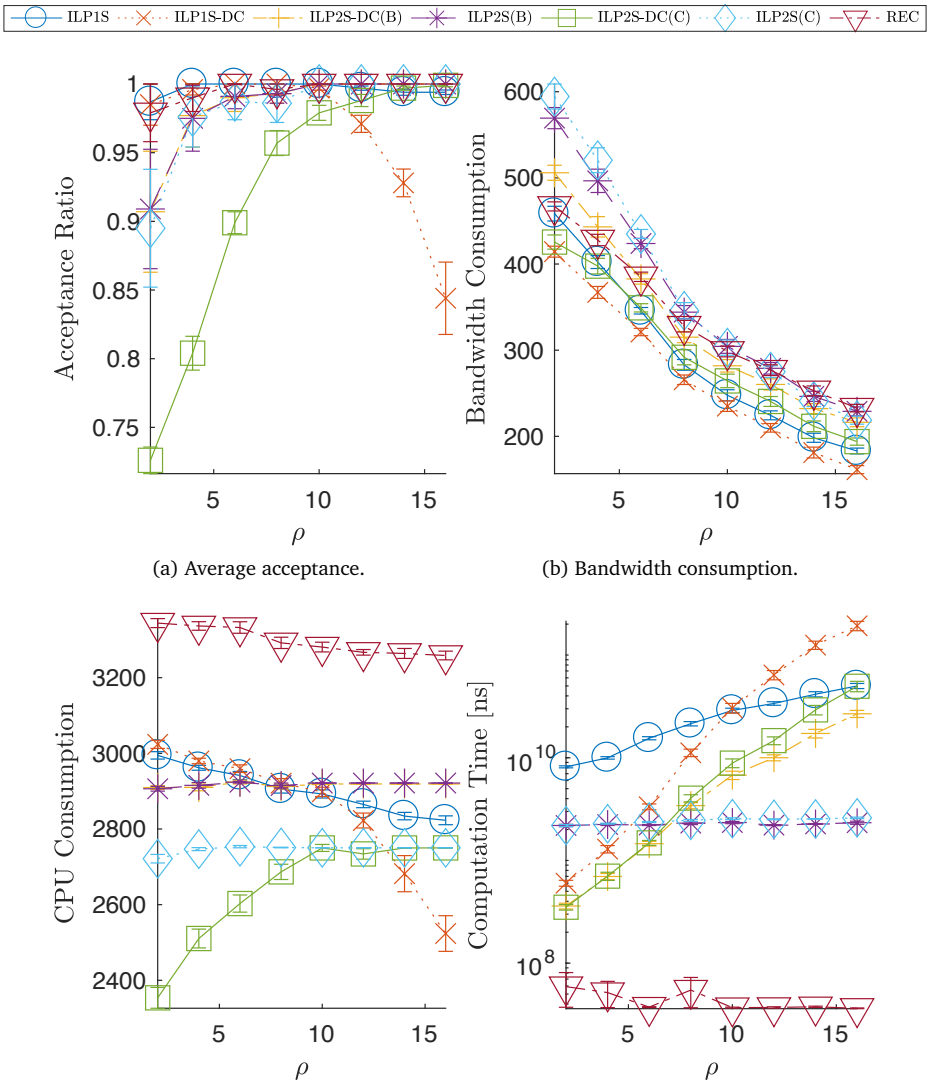
(a) Average acceptance.

(b) Bandwidth consumption.

(c) CPU consumption.

(d) Average computation time.

Figure 6.5: Influence of the PL bandwidth capability B_{avg}.



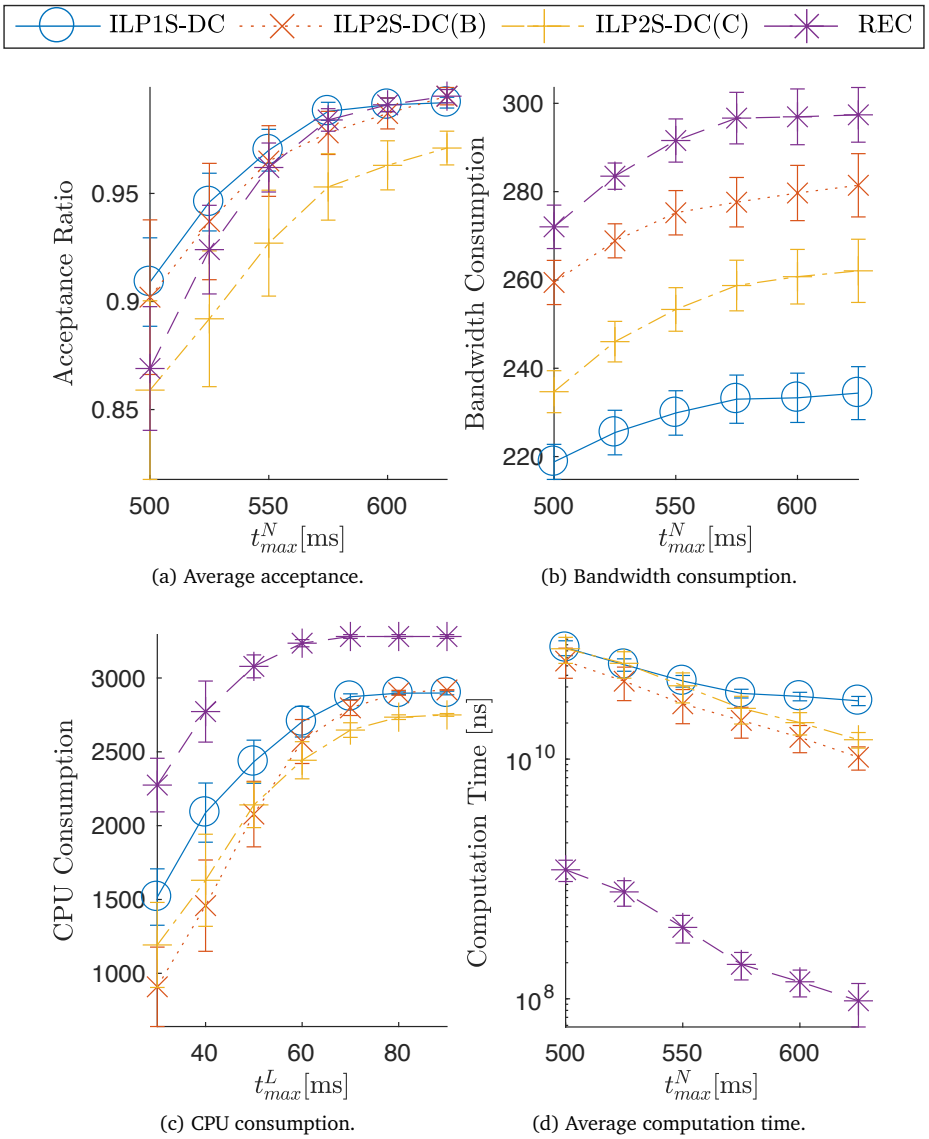
(a) Average acceptance.

(b) Bandwidth consumption.

(c) CPU consumption.

(d) Average computation time.

Figure 6.6: Influence of the radius (ρ) for $B_{avg} = 6000$.



(a) Average acceptance.

(b) Bandwidth consumption.

(c) CPU consumption.

(d) Average computation time.

Figure 6.7: Influence of the maximum end-to-end latency for $B_{avg} = 6000$ and $\rho = 10$.

Figure 6.6 shows the influence of the radius ρ. As ρ increases, the number of candidates for each VNF, except for source and target VNFs, increases. The acceptance ratio for REC is very close to that of ILP1S (within 1%). The performances of ILP2S(C), ILP2S(B) and ILP2S-DC(B) are very similar, all have an acceptance ratio about 9% lower than for ILP1S. As ρ increases the computation time increases strongly for the path-based ILPs. The computation time for ILP1S-DC often exceeds the maximum allowed time of 10 minutes, causing the acceptance ratio to drop for ρ exceeding 8. Observe that for higher ρ values ILP2S-DC(C) can place all requests, the CPU consumption then converges to that for ILP2S(C). The acceptance ratio for REC is up to 35% higher than for ILP2S(C), while requiring up to 1376× less computation time.

Figure 6.7 shows the impact of the maximum end-to-end delay t^N_{max}. As this maximum delay increases the acceptance ratio and required computation time increases for all algorithms. The acceptance ratio for REC can slightly exceed that of ILP1S-DC since not all paths between any two PMs are considered. The bandwidth and CPU consumption is the highest for REC. The acceptance ratio is highest for ILP1S-DC, it is up to 6% higher than for the 2-stage ILPs, while requiring only 3× more computation time. For very strict delay constraints, ILP2S-DC(B) outperforms REC, but for maximum delays of 575 ms and higher, REC outperforms both 2-phase algorithms. The performance for strict delay-constraints is determined by its backtracking limit. A higher backtracking limit can improve the acceptance ratio but comes at the cost of an increased worst-case computation time. The performance of REC is up to 4% lower than for ILP1S-DC, however it can find a solution up to 337× faster. The acceptance ratio of ILP2S-DC(C) is the lowest.

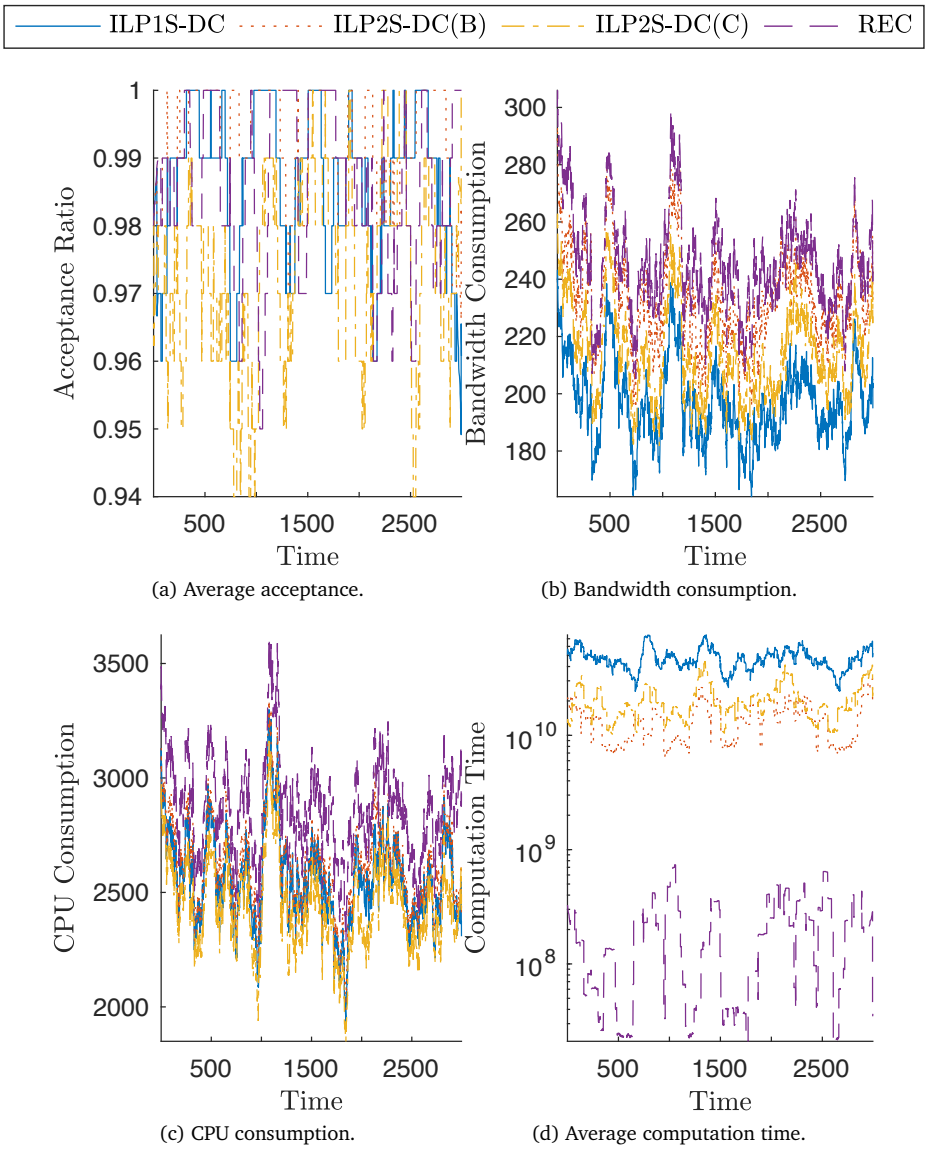
Online

The online traces for a maximum end-to-end delay of 575 ms are shown in Figure 6.8.

In total 3000 requests are processed sequentially. In the online scenario, the traces are smoothed by a moving average with a window of size 100 requests. In these traces the acceptance ratio of ILP1S-DC and REC is the highest. The acceptance ratio for ILP2S-DC(B) is a few % lower. It is the lowest for ILP2S-DC(C). The SN bandwidth consumption per accepted request is the lowest for ILP1S-DC. The CPU consumption is the lowest for ILP2S-DC(C) since this algorithm minimizes the CPU consumption in the composition phase and the CPU consumption is independent of the embedding. The execution time is the lowest for REC, its computation time shows a large spread. When finding a feasible solution is easy, the algorithm finishes fast. If finding a solution is more difficult, the algorithm performs more backtracks, resulting in an increased computation time. Ultimately, its computation time is limited by the backtracking limit and the number of neighbors considered in each recursion step. The execution time for the 2-phase ILPs is significantly higher than for REC, the execution time for the 1-stage ILP is the highest.

6.5.6 Conclusions

In this section, we focused on the resource allocation challenges related to the orchestration of NSs in NFV environments. We introduced a service model with improved applicability, that can



(a) Average acceptance.

(b) Bandwidth consumption.

(c) CPU consumption.

(d) Average computation time.

Figure 6.8: Online traces for $B_{avg} = 6000$, $\rho = 10$ and $t^N_{max} = 575\,\text{ms}$.

describe a richer class of VNF-FGs, while considering the NS delay constrains. We proposed an optimal placement algorithm that can adapt the VNF-FG to the availability of resources in the SNe, while considering the SRs. We demonstrated that our optimal algorithm can improve the acceptance ratio by up to 42% while requiring only 11 times more computation time, compared to algorithms that do not coordinate composition and embedding. Our proposed approach can find the best PC for any cloud environment. Further, we proposed a heuristic that can find near-optimal solutions that are within 1% of the optimum. Through coordination of the composition and embedding, the heuristic offers a 35% improvement in acceptance ratio, compared to uncoordinated approaches that minimize CPU consumption in the first stage, while being up to 1376 times faster. Clearly, our approach can significantly improve the provider revenue for a wide range of service orchestration scenarios, through better coordination of the service composition and embedding tasks.

Chapter 7

Conclusions

This chapter sums up the main contributions and results of the thesis that were mainly directed to investigating the research questions formulated in Section 1.3, in conjunction with the exploitation of the research opportunities identified in Section 2.4.7.

Section 7.1 details the main results obtained throughout the development of the thesis. Section 7.2 shows the participation of the author of this thesis in national and international research projects and how the contributions of this thesis were developed following the main objectives of these projects.

7.1 Main contributions and results

Chapter 3 investigates how to effectively replicate data across storage nodes in heterogeneous cloud environments, as asked by Question I. We approach this problem as a runtime revenue optimization problem, i.e., the GRMP. This problem is the task to periodically decide on the optimal RL of a replicated stateful (data-)service over its required lifetime, while considering at the same time the current replication state, the distribution of failures and recovery times, the replication scheme and the agreed durability SLA. To maximize the provider revenue (Subquestion Ia), we propose an exact algorithm, based on DP, which considers both a dynamic replication model and SLAs regarding durability. Data loss (Subquestion Ib) and the impact of management decisions can be modeled by extending the existing availability model proposed by Google researchers [42]. This existing Markov model at the same time considers correlated failure, the recovery time distribution and operational costs. We define additional states that correspond to the decision to scale in or out a service, while waiting for these decisions to take effect. Next, we investigate how the storage SLA impacts data-availability in the optimal replication strategy (Subquestion Ic). Through extensive simulation results, we demonstrated that the probability of data loss decreases as the provided reward for successful storage of the data for the entire desired lifetime goes up. When the reward goes up, the provider should incur greater storage and recovery costs to maximize his expected pay-off. Further, we demonstrate that given a fixed reward per unit time, the expected pay-off for the provider initially increases as the request duration goes up. However, as the request duration further increases, the probability of data loss increases too. Eventually, the expected

provider revenue decreases towards the agreed penalty for data loss. The cloud characteristics (Subquestion Id) impact the optimal replication strategy in the following way. First, a reduced failure probability, corresponding to an increased MTTF, calls for a reduced RL. When the failure probability decreases, minimizing the operational costs becomes more important than minimizing the probability of data loss. Similarly, when the operational costs increase, the optimal RL decreases. Further, an increased correlation between storage node failures increases the size of a failure burst, which makes it harder to protect against data loss. Finally, as per Subquestion Ie, we compare how our proposed replication strategy performs, compared to traditional replication strategies. We demonstrate that our proposed approach can significantly improve provider revenue, compared to existing management algorithms. These existing algorithms either always try to immediately restore the RL to a predefined value, or restore the RL to its maximum value only when a minimum number of failures has been detected. In both time-invariant and time-variant cloud environments, our algorithm can adapt the RL to varying system parameters and maximize the expected reward.

Chapter 4 investigates how to protect NSs against node and link failure in heterogeneous cloud environments, as per Question II. To deal with the scarcity of resources at the edge of the network (Subquestion IIa), we propose an approach that avoids wasting resources on applications whose availability requirements cannot be met. Further, due to the reliability spread in the geo-distributed infrastructure, coupled with the resource scarcity at the edge and the resource overhead associated with protections in the PC, we expect approaches that introduce a fixed protection level across the entire infrastructure to perform poorly. Therefore, we propose an approach that introduces protections only, where they are needed. Through the introduction of duplicates, we can realize very fine-grained application-level protections for stateless applications (Subquestion IIb). Each duplicate is a separate instance of the application's VNe. Bandwidth, memory and processing resources can be shared between duplicates of the same application, since these applications are stateless and only one of the duplicates must be active at the same time. We formulate the problem of placing services with minimal cost, while guaranteeing a minimum availability for each application as an optimization problem, i.e. the availability-aware SVNE problem. To guarantee a minimum availability for an application (Subquestion IIc), an availability-model is needed that can calculate the expected end-to-end availability for each valid PC, based on the failure distribution of PMs and PLs. In our proposed availability-model (Subquestion IId), an application is available if at least 1 of its duplicates is available. Our proposed model considers the correlation between the failures of any two duplicates. Hence, our model does not require copies of the VNe to be placed on disjoint subgraphs of the SNe, which is a common requirements of traditional protection mechanisms. Finally, we compare the performance of our proposed algorithms to traditional strategies (Subquestion IIe). Our ILP formulation of the availability-aware SVNE problem, can be used to find an exact solution with minimal cost for smaller problem instances. Compared to approaches that are not availability-aware or that introduce a fixed level of protection, it can significantly reduce the placement cost and improve the number of application requests that can be accepted. Due to the computational hardness of the problem, finding exact solutions for larger problem instances is intractable. Therefore, we propose a distributed fault-tolerant meta-heuristic based on GA: a distributed set of workers concurrently search for the best SVNE. Additionally, we present a scalable centralized algorithm using the

paradigm of subgraph isomorphism: this heuristic approach provides ultra-fast placement of applications at the expense of reduced optimality.

Chapter 5 focuses on research Question III. It investigates how to orchestrate NSs in geo-distributed NFV environments. This task requires composition of the NS's VNF-FG based on its SRs and embedding of the composed VNF-FG on the SNe. In this chapter, the composition of the VNFs is based on an existing service model that can generate VNF-FGs with a directed tree topology (Subquestion IIIa). This service topology is applicable to services where content is provided and distributed by the content provider. In this model, the NS must originate in a single root VNF instance and terminate in a set of terminal VNF instances. The VNFs that connect the root VNF instance to the terminal VNF instances can each only have a single ingress VL instance, but possibly multiple egress VL instances. This composition model considers precedence and compatibility constraints in the chaining, to restrict the sequence in which the traffic can flow through the VL instances. Geo-distributed VNF environments are highly heterogeneous, because of a combination of LCs, delay constraints and connectivity limitations. Therefore, it is expected that the consideration of the SNe during VNF-FG composition can significantly ease the embedding and improve the resulting placement quality. Therefore, this chapter proposes two algorithms that can coordinate the composition and embedding tasks (Subquestion IIIb). First, we formulate the combined composition and embedding problem to find a minimum-cost PC as an ILP that can be used to find an exact solution for smaller problem instances. Second, we propose a fast, greedy heuristic that iteratively selects an SFC to add to the VNF-FG and embeds it at the same time. Finally, we compare the performance of the proposed algorithms that can coordinate the composition and embedding tasks to algorithms that perform these two tasks in separate stages (Subquestion IIIc). Through extensive performance evaluations, we demonstrate that approaches that coordinate both tasks can significantly improve the number of requests that can be accepted at the same time and reduce the placement cost. More stringent LCs increase the heterogeneity of the environment and cause the performance gap between coordinated and uncoordinated approaches to increase. Increasing the flexibility with which the VNFs can be chained has a similar effect. Further, when the workload increases, resources become scarcer and the acceptance decreases for all approaches. However, the acceptance ratio of coordinated approaches can be up to 15% higher than for uncoordinated approaches. The proposed greedy heuristic can find near-optimal solutions.

Compared to Chapter 5, Chapter 6 proposes a novel service model with improved applicability, together with the required resource allocation algorithms to orchestrate such services. It focuses on Subquestion IIId, i.e., how to orchestrate NSs in NFV environments that require traffic aggregation. The proposed service model can produce DAG VNF-FGs with support for optional performance enhancing VNFs, bidirectional chaining constraints (e.g., for heterogeneous service delivery), traffic aggregation (e.g., fusion of sensor data in WSNs or composition of multiple content sources) and end-to-end delay constraints. To deal with the computational intractability associated with finding an exact solution (Subquestion IIa), we propose a recursive heuristic that in each step adds a candidate VL instance to the composition and embeds it at the same time. Again, we demonstrate that through coordination of the composition and embedding tasks, the number of accepted applications and the placement cost can be improved significantly over a wide range of cloud conditions and SRs.

7.2 Research projects

The research contributions made by this thesis are made in the context of several research projects. The author has actively participated in the set of projects summarized in Table 7.1. States 'F' and 'E' indicate that at the time of writing the project is 'Finished' or 'Executing', respectively.

Table 7.1: Research projects.

Project name	Funding	Duration	State	Industry partners
COST ACROSS	European	14/11/2013 - 13/11/2017	F	TNO (Netherlands), Deutsche Telekom AG
iFEST	Flemish	31/12/2014 - 30/12/2016	F	ID&T, Playpass, Sentiance, Sendrato, Telenet, 3factr
EMD	Flemish	01/01/2015 - 31/12/2016	F	IMTECH ICT, SDNsquare, Barco
5GUARDS	Flemish	01/04/2017 - 31/03/2019	E	Accelleran, Ericsson, Orange, Rombit
FUSE	Flemish	01/10/2017 - 30/09/2019	E	E-BO Enterprises, Axians, Barco

Autonomous Control for a Reliable Internet of Services (ACROSS) is a project funded by European Cooperation in Science & Technology (COST) that focuses on the resource allocation challenges related to the paradigm shift from the traditional information-oriented Internet into an Internet of Services (IoS). These challenged are related to the deployment of large-scale service chains, combining and integrating the functionality of huge numbers of other services offered by third parties, including cloud services. The aim of this project is to create a European network of expertise, from both academia and industry, aiming at the development of autonomous control methods and algorithms for a reliable and quality-aware IoS. Within the framework of this project, the author of this thesis performed a research stay of 1 month under supervision of Prof. Rob van der Mei at the Stochastics department at CWI.

The improved Festival Experience with wearable Sensor Technology (iFEST) project is realized in collaboration with imec, with project support from Flanders Innovation & Entrepreneurship (VLAIO). The goal of the iFEST project is to improve the digital experience of large events, such as music festivals, by developing a new generation of festival bracelets. The author actively collaborated on addressing the infrastructural and computational challenges that arise in setting up festival applications. These challenges are three-fold. First, as festivals are typically located on geographically remote areas, there is little up-link connectivity or computational infrastructure available nearby, resulting in connectivity issues. Second, the infrastructural capabilities that festival locations offer are diverse. Small festivals may not have the ability to invest in local infrastructure (and thus mainly rely on public cloud offerings), while larger festivals often already have on-site infrastructure available. As the iFEST platform needs to be portable, i.e., reusable on several festivals, it needs to address these heterogeneous infrastructures. The proposed resource allocation algorithms to orchestrate reliable NSs in

these heterogeneous environments are presented in Chapter 4.

Elastic Media Distribution for online collaboration (EMD) is a project realized in collaboration with imec, with project support from VLAIO. EMD pursues the creation of a flexible, scalable and reliable cloud-based platform for the realtime and secure distribution of audio and video content. In order to realize reliable NSs on top of best-effort computing and networking environments, subject to both SNo- and SL-failure, availability-aware SVNE resource allocation algorithms are needed. These algorithms are developed in Chapter 4.

5G qUAlity slicing foR the Deployment of Security services (5GUARDS) is a project realized in collaboration with imec, with project support from VLAIO. It investigates the slicing capability of 5G that enables dynamic resource allocation to match demand. 5GUARDS explores how multiple security services with a wide range of requirements can be supported simultaneously by 5G networks. Within the framework of the 5GUARDS project, the author of this thesis, developed the NFV service models proposed in Chapters 5 and 6; and the corresponding resource allocation algorithms to support on-demand orchestration.

Flexible federated Unified Service Environment (FUSE) is a project realized in collaboration with imec, with project support from VLAIO. It addresses the challenges related to setting up cross-organizational collaborations between software platforms operating in isolated domains. FUSE develops a single service architecture that securely connects many software components and domains flexibly and dynamically on a temporary basis. Within the framework of the FUSE project, the author of this monograph developed the required network provisioning algorithms to orchestrate the requested NSs and the algorithms to perform outcome-based service optimization. The ideas for the network provisioning algorithms are presented in Chapters 5 and 6. The developed algorithms to optimize the RL of replicated services in order to maximize the provider revenue subject to SLAs regarding data loss, are presented in Chapter 3.

Chapter 8

Future work

The contributions presented in this thesis can be used in several existing network virtualization projects and testbeds with management systems that use orchestration algorithms to automatically allocated service requests to SNe resources. The contributions introduced throughout this thesis can still be improved, especially with regard to applicability. This chapter is divided into two sections: Section 8.1 summarizes the possible scenarios where the contributions introduced in this thesis can be applied. Section 8.2 details the possible enhancements that can be made in order to evolve and improve the current state of the presented contributions.

8.1 Applicability of thesis contributions

The algorithms proposed in Chapter 3 decide on the RL that optimizes the provider revenue, over a service's lifetime. The approach is based on a dynamic failure model considering the impact of correlated failures, time between failures and recovery times. These algorithms can be applied to optimize the RL for very large-scale datacenters that have symmetry in their structure, e.g. the failure behavior of nodes within a rack is homogeneous. An increased scale results in more accurate estimation of the failure and replication parameters. Further, the complexity of the solution only depends on the maximum RL and the length of a MI, not on the dimensions of the SNe. When combined with monitoring data from a cloud monitoring platform, e.g., Prometheus [116], and a homogeneous SNe, e.g. a single rack, the algorithms can be readily applied to decide on the optimal RL of a replicated file system, e.g. HDFS.

The availability-aware SVNE algorithms can be used to orchestrate stateless NSs, e.g. video streaming services, that have strict QoS requirements. The availability-model does require accurate estimates on the availability distribution of the SNe components. To the best of our knowledge, there are no practical VIM frameworks that support SVNE. Current VIMs, e.g., OpenStack [117], can protect against VNo failures, but not against VL failures.

The NFV service models presented in Chapters 5 and 6 can be used to orchestrate practical NSs. Currently, there is no practical data model that supports partial ordering of VNFs in an SFC. For instance, the TOCSA [30] model assumes a total ordering of the VNFs. The proposed algorithms can be used to orchestrate services that adhere to the TOCSA model, however in this scenario our algorithms cannot exploit any flexibility in the chaining. An extension of this

model to support partial orderings, would allow this optimization.

8.2 Further improvement of the contributions

The contributions presented in this thesis can be further evolved and improved along the following directions.

8.2.1 Application to 5G network slicing

The orchestration algorithms presented in this monograph generally assume that the controller has full control over the routing of traffic flows through the SNe. It is assumed that this controller can freely determine the PP in the SNe over which the traffic flows and that it can reserve bandwidth on each PL along the PP. This assumption holds true in a network where all interconnections are wired and the SDN-enabled switches can be configured by the controller. Network slicing, which will be a key feature of the 5G standard, promises network operators the capability to create multiple VNes on top of a common, shared physical infrastructure. Partitioning of the network into VNes that can be controller using software will enable providers to optimize each network slice for a specific objective, e.g., availability, throughput or latency. The core network is mainly wired and can largely be controlled using SDN. In contrast, the RAN comprises wireless resources, which are much harder to manage. Here, interference among nearby User Equipment (UE) and basestations makes it difficult to provide performance guarantees. Depending on the employed wireless technologies, frequency-; timeslot-; and code-reservations must be carefully coordinated between nearby basestations. Dynamically provisioning these resources, to adapt to changing workload and networking conditions, while limiting the management overhead to an acceptable level, is very challenging. Therefore, future research is needed into management algorithms that can enable performance isolation between network slices that coexist at the RAN and to enable end-to-end orchestrations across the RAN and core network.

8.2.2 Decentralized approach

The management approaches presented in this thesis assumed a single centralized controller that is in charge of the entire cloud management. A centralized controller ultimately limits the scalability of the cloud environment. This controller requires a complete view on the resources available in the infrastructure and requires all relevant information to be transferred to a single location, causing a significant management overhead. Further, a single centralized controller presents a single point of failure. When the centralized controller fails, it will stop the entire system from working. Clearly, a single point of failure is highly undesirable. Therefore, further research is needed to develop the required resource allocation algorithms to implement such a distributed controller, e.g., a hierarchical controller. A major challenge in this regard is to limit the optimality gap between a centralized and a distributed controller.

8.2.3 Practical realization through protocols

The orchestration algorithms developed in the monograph implement the resource allocation functions, to be carried out by a single centralized management entity. As discussed above, a centralized controller is undesirable for a reliable practical implementation. Instead, distributed control is needed. These protocols regulate both resource- and service-discovery and resource allocation. The design of these protocols is challenging as both the correct behavior of these protocols by themselves and the interoperability with other protocols must be guaranteed. To enable distributed control, further research is needed into the specification, testing and verification of these supporting protocols.

Contributions

[C1] **B. Spinnewyn**, R. Mennes, J. Botero, and S. Latré, "Resilient application placement for geo-distributed cloud networks," *Journal of Network and Computer Applications*, vol. 85, pp. 14–31, 2017.

[C2] **B. Spinnewyn**, P. H. Isolani, C. Donato, J. F. Botero, and S. Latré, "Coordinated service composition and embedding of 5g location-constrained network functions," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2018.

[C3] **B. Spinnewyn**, J. F. Botero, and S. Latré, "Delay-constrained nfv orchestration for heterogeneous cloud networks," 2019, submitted to IEEE/ACM Transactions on Networking.

[C4] **B. Spinnewyn** and S. Latré, "Towards a fluid cloud: an extension of the cloud into the local network," in *IFIP International Conference on Autonomous Infrastructure, Management and Security*. Springer, 2015, pp. 61–65.

[C5] **B. Spinnewyn**, B. Braem, and S. Latré, "Fault-tolerant application placement in heterogeneous cloud environments," in *2015 11th International Conference on Network and Service Management (CNSM)*, Nov 2015, pp. 192–200.

[C6] **B. Spinnewyn**, J. Botero, and S. Latré, "Cost-effective replica management in fault-tolerant cloud environments," in *Network and Service Management (CNSM), 2017 13th International Conference on*. IEEE, 2017, pp. 1–9.

[C7] **B. Spinnewyn**, C. Donato, J. F. Botero, and S. Latré, "Effective nfv orchestration for wide-ranging services across heterogeneous cloud networks," in *Proceedings of IM 2019 (accepted for publication)*. IEEE, nov 2019.

[C8] R. Mennes, **B. Spinnewyn**, S. Latré, and J. F. Botero, "Greco: A distributed genetic algorithm for reliable application placement in hybrid clouds," in *2016 5th IEEE International Conference on Cloud Networking (CLOUDNET)*, Oct. 2016, pp. 14–20.

Bibliography

[1] J. Markoff, "Building the electronic superhighway," *The New York Times*, Jan 1993.

[2] "Dropbox," https://www.dropbox.com/?landing=dbv2, 2018, (Accessed on 11/29/2018).

[3] "Muziek voor iedereen - spotify," https://www.spotify.com/be-nl/, 2019, (Accessed on 01/16/2019).

[4] "Youtube tv - watch & dvr live sports, shows & news," https://tv.youtube.com/welcome/, 2018, (Accessed on 09/10/2018).

[5] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862 – 876, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128609003387

[6] B. A. A. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys Tutorials*, vol. 16, no. 3, pp. 1617–1634, Third 2014.

[7] C. Tankard, "What the gdpr means for businesses," *Network Security*, vol. 2016, no. 6, pp. 5 – 8, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1353485816300563

[8] "Netflix belgië - kijk series online, kijk films online," https://www.netflix.com/be/, 2019, (Accessed on 01/16/2019).

[9] R. Jimenez, "Integrating smartphones in spotify's peer-assisted music streaming service," PhD dissertation, School of Information and Communication Technology (ICT), Communication Systems, 2013.

[10] "How spotify migrated everything from on-premise to google cloud platform - computerworld," https://www.computerworld.com.au/article/644574/how-spotify-migrated-everything-from-on-premise-google-cloud-platform/, 2019, (Accessed on 01/17/2019).

[11] "Twitch," https://www.twitch.tv/, 2018, (Accessed on 09/10/2018).

[12] C. Zhang and J. Liu, "On crowdsourced interactive live streaming: a twitch. tv-based measurement study," in *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, 2015, pp. 55–60.

[13] J. Deng, G. Tyson, F. Cuadrado, and S. Uhlig, "Internet scale user-generated live video streaming: The twitch case," in *International Conference on Passive and Active Network Measurement*. Springer, 2017, pp. 60–71.

[14] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE access*, vol. 6, pp. 6900–6919, 2018.

[15] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[16] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, vol. 23, no. 3, pp. 567–619, 2015.

[17] "G suite - gmail, google documenten, drive, agenda en meer voor bedrijven," https://gsuite.google.com/, 2018, (Accessed on 11/29/2018).

[18] "Online meeting software with hd video conferencing | gotomeeting," https://www.gotomeeting.com/, 2018, (Accessed on 11/29/2018).

[19] "What is aws? - amazon web services," https://aws.amazon.com/what-is-aws/, 2018, (Accessed on 11/29/2018).

[20] "Rackspace: Managed dedicated & cloud computing services," https://www.rackspace.com/, 2018, (Accessed on 11/29/2018).

[21] "Compute engine - iaas | compute engine | google cloud," https://cloud.google.com/compute/, 2018, (Accessed on 11/29/2018).

[22] "Apache openwhisk is a serverless, open source cloud platform," https://openwhisk.apache.org/, 2018, (Accessed on 11/29/2018).

[23] "Openlambda," https://open-lambda.org/, 2018, (Accessed on 11/29/2018).

[24] "Azure functionsâĂŤserverless architecture | microsoft azure," https://azure.microsoft.com/en-us/services/functions/, 2018, (Accessed on 11/29/2018).

[25] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[26] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[27] J. Al-Muhtadi, R. Campbell, A. Kapadia, M. D. Mickunas, and S. Yi, "Routing through the mist: privacy preserving communication in ubiquitous computing environments," in *Proceedings 22nd International Conference on Distributed Computing Systems*, 2002, pp. 74–83. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1022244

[28] G. Hu, W. P. Tay, and Y. Wen, "Cloud robotics: Architecture, challenges and applications," *IEEE Network*, vol. 26, no. 3, pp. 21–28, may 2012.

[29] N. Feamster, L. Gao, and J. Rexford, "How to lease the internet in your spare time," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 1, pp. 61–64, 2007.

[30] D. P. Paul Lipton, John Crandall, "Instance model for tosca version 1.0," http://docs.oasis-open.org/tosca/TOSCA-Instance-Model/v1.0/TOSCA-Instance-Model-v1.0.html, 2017, (Accessed on 01/10/2019).

[31] "List of vnfs - functest - opnfv wiki," https://wiki.opnfv.org/display/functest/List+Of+VNFs, 2019, (Accessed on 01/24/2019).

[32] B. Carpenter and S. Brim, "Middleboxes: Taxonomy and issues," IBM Zurich Research Laboratory, Tech. Rep. RFC 3234, 2002.

[33] B. Yi, X. Wang, K. Li, M. Huang *et al.*, "A comprehensive survey of network function virtualization," *Computer Networks*, 2018.

[34] J. C. Linsey Miller, Nagesh Puppala, "Virtual video transcoding in the cloud," Artesyn Embedded Technologies, Intel, Dell, Tech. Rep., 04 2017.

[35] Y. Li and M. Chen, "Software-defined network function virtualization: A survey," *IEEE Access*, vol. 3, pp. 2542–2553, 2015.

[36] C. Somerville, "Small cell virtualization functional splits and use cases," Small Cell Forum, Tech. Rep. SFC159, 01 2016.

[37] M. Ersue, "Etsi nfv management and orchestration-an overview," in *Proc. of 88th IETF meeting*, 2013.

[38] J. Puchinger, G. R. Raidl, and U. Pferschy, "The multidimensional knapsack problem: Structure and algorithms," *INFORMS Journal on Computing*, vol. 22, no. 2, pp. 250–265, 2010.

[39] ISO/IEC-25010, "Systems and software engineering âĂŞ Systems and software Quality Requirements and Evaluation (SQuaRE) âĂŞ System and software quality models." International Organization for Standardization, Geneva, CH, Standard, mar 2010.

[40] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible, "Improving performance and availability of services hosted on iaas clouds with structural constraint-aware virtual machine placement," in *Services Computing (SCC), 2011 IEEE International Conference on*. IEEE, 2011, pp. 72–79.

[41] W. Wang, H. Chen, and X. Chen, "An availability-aware virtual machine placement approach for dynamic scaling of cloud applications," in *Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing (UIC/ATC), 2012 9th International Conference on*. IEEE, 2012, pp. 509–516.

[42] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems." in *OSDI*, 2010, pp. 61–74.

[43] M. Silberstein, L. Ganesh, Y. Wang, L. Alvisi, and M. Dahlin, "Lazy means smart: Reducing repair bandwidth costs in erasure-coded distributed storage," in *Proceedings of International Conference on Systems and Storage*. ACM, 2014, pp. 1–7.

[44] S. Nath, H. Yu, P. B. Gibbons, and S. Seshan, "Subtleties in tolerating correlated failures in wide-area storage systems." in *Proceedings of the 3rd Symposium on Networked Systems Design & Implementation*, 2006.

[45] S. Ramabhadran and J. Pasquale, "Analysis of long-running replicated systems." in *INFOCOM*, vol. 2006, 2006, pp. 1–9.

[46] H. Weatherspoon and J. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, ser. IPTPS '01. London, UK, UK: Springer-Verlag, 2002, pp. 328–338. [Online]. Available: http://dl.acm.org/citation.cfm?id=646334.687814

[47] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 1888–1906, Fourth 2013.

[48] E. Amaldi, S. Coniglio, A. Koster, and M. Tieves, "On the computational complexity of the virtual network embedding problem," *Electronic Notes in Discrete Mathematics*, vol. 52, pp. 213–220, 06 2016.

[49] N. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *INFOCOM 2009, IEEE*. IEEE, 2009, pp. 783–791.

[50] M. Melo, S. Sargento, U. Killat, A. Timm-Giel, and J. Carapinha, "Optimal virtual network embedding: Node-link formulation," *IEEE Transactions on Network and Service Management*, vol. 10, no. 4, pp. 356–368, 2013.

[51] A. Razzaq and M. S. Rathore, "An approach towards resource efficient virtual network embedding," in *Evolving Internet (INTERNET), 2010 Second International Conference on*. IEEE, 2010, pp. 68–73.

[52] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, "Virtual network embedding through topology-aware node ranking," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 2, p. 38, 2011.

[53] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," in *Proc. of the 1st ACM workshop on Virtualized infrastructure systems and architectures*. ACM, 2009.

[54] S. Herker, A. Khan, and X. An, "Survey on Survivable Virtual Network Embedding Problem and Solutions," in *Proceedings of ICNS 2013, The Ninth International Conference on Networking and Services*, mar 2013, pp. 99–104.

[55] M. R. Rahman and R. Boutaba, "SVNE: Survivable virtual network embedding algorithms for network virtualization," *IEEE Transactions on Network and Service Management*, vol. 10, no. 2, pp. 105–118, 2013.

[56] M. M. A. Khan, N. Shahriar, R. Ahmed, and R. Boutaba, "SiMPLE: Survivability in multi-path link embedding," in *Proceedings of the 11th International Conference on Network and Service Management, CNSM 2015*, 2015, pp. 210–218.

[57] S. Chowdhury, R. Ahmed, M. M. ALAM KHAN, N. Shahriar, R. Boutaba, J. Mitra, and F. Zeng, "Dedicated Protection for Survivable Virtual Network Embedding," in *IEEE Transactions on Network and Service Management*, 2016, pp. 1–1. [Online]. Available: http://ieeexplore.ieee.org/document/7480798/

[58] G. A. WG, "5g architecture white paper," 5GPPP Architecture Working Group, Tech. Rep., 12 2017.

[59] J. G. Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey," *IEEE TNSM*, vol. 13, no. 3, pp. 518–532, Sept 2016.

[60] Y. C. Lee and A. Y. Zomaya, "Energy efficient utilization of resources in cloud computing systems," *The Journal of Supercomputing*, vol. 60, no. 2, pp. 268–280, 2012.

[61] Z. Zhong, K. Chen, X. Zhai, and S. Zhou, "Virtual machine-based task scheduling algorithm in a cloud computing environment," *Tsinghua Science and Technology*, vol. 21, no. 6, pp. 660–667, Dec 2016.

[62] R. Camati, A. Calsavara, and L. L. Jr, "Solving the Virtual Machine Placement Problem as a Multiple Multidimensional Knapsack Problem," in *ICN 2014, The Thirteenth ...*, 2014, pp. 253–260. [Online]. Available: http://www.thinkmind.org/index.php?view=article{\&}articleid=icn{_}2014{_}11{_}10{_}30065

[63] A. Jarray and A. Karmouch, "Decomposition approaches for virtual network embedding with one-shot node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 23, no. 3, pp. 1012–1025, Jun. 2015. [Online]. Available: https://doi.org/10.1109/TNET.2014.2312928

[64] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *Network and Service Manage. (CNSM), 2015 11th Int. Conf. on*. IEEE, 2015, pp. 50–56.

[65] N. Bouten, J. Famaey, R. Mijumbi, B. Naudts, J. Serrat, S. Latré, and F. D. Turck, "Towards nfv-based multimedia delivery," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 738–741.

[66] A. F. Ocampo, J. Gil-Herrera, P. H. Isolani, M. C. Neves, J. F. Botero, S. Latré, L. Zambenedetti, M. P. Barcellos, and L. P. Gaspary, "Optimal service function chain composition in network functions virtualization," in *IFIP International Conference on Autonomous Infrastructure, Management and Security*. Springer, 2017, pp. 62–76.

[67] M. Rost and S. Schmid, "Service chain and virtual network embeddings: Approximations using randomized rounding," *CoRR*, vol. abs/1604.02180, 2016. [Online]. Available: http://arxiv.org/abs/1604.02180

[68] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou, "Traffic aware placement of interdependent nfv middleboxes," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017, pp. 1–9.

[69] O. Houidi, O. Soualah, W. Louati, M. Mechtri, D. Zeghlache, and F. Kamoun, "An efficient algorithm for virtual network function scaling," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, Dec 2017, pp. 1–7.

[70] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers," *Future Generation Computer Systems*, vol. 28, no. 2, pp. 358–367, 2012.

[71] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.

[72] R. N. Calheiros, R. Buyya, and C. A. F. D. Rose, "A heuristic for mapping virtual machines and links in emulation testbeds," in *2009 International Conference on Parallel Processing*, Sep. 2009, pp. 518–525.

[73] G. Somani, P. Khandelwal, and K. Phatnani, "Vupic: Virtual machine usage based placement in iaas cloud," *arXiv preprint arXiv:1212.0085*, 2012.

[74] H. Jiang, Y. Wang, L. Gong, and Z. Zhu, "Availability-aware survivable virtual network embedding in optical datacenter networks," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 7, no. 12, pp. 1160–1171, 2015.

[75] K. Inoue, S. Arakawa, S. Imai, T. Katagiri, and M. Murata, "Noise-induced vne method for software-defined infrastructure with uncertain delay behaviors," *Computer Networks*, vol. 145, pp. 118–127, 2018.

[76] A. Marotta and A. J. Kassler, "A power efficient and robust virtual network functions placement problem," in *28th International Teletraffic Congress (ITC 28)*, WÃijrzburg, Germany, 2016.

[77] C. Ghribi, M. Mechtri, O. Soualah, and D. Zeghlache, "Sfc provisioning over nfv enabled clouds," in *2017 IEEE 10th Int. Conf. on Cloud Computing (CLOUD)*, June 2017, pp. 423–430.

[78] L. Gong, H. Jiang, Y. Wang, and Z. Zhu, "Novel location-constrained virtual network embedding lc-vne algorithms towards integrated node and link mapping," *IEEE/ACM Trans. on Networking*, vol. 24, no. 6, pp. 3648–3661, December 2016.

[79] S. Haeri and L. TrajkoviÄĞ, "Virtual network embedding via monte carlo tree search," *IEEE Transactions on Cybernetics*, vol. 48, no. 2, pp. 510–521, Feb 2018.

[80] T. Wang, X. Lu, and M. Hou, "Novel algorithm for distributed replicas management based on dynamic programming," *Journal of Systems Engineering and Electronics*, vol. 17, no. 3, pp. 669–672, 2006.

[81] B. Meroufel and G. Belalem, "Managing data replication and placement based on availability," *AASRI Procedia*, vol. 5, pp. 147–155, 2013.

[82] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 98–106.

[83] T. Lukovszki and S. Schmid, "Online admission control and embedding of service chains," in *Int. Colloquium on Structural Information and Communication Complexity*. Springer, 2015, pp. 104–118.

[84] Z. Allybokus, N. Perrot, J. Leguay, L. Maggi, and E. Gourdin, "Virtual function placement for service chaining with partial orders and anti-affinity rules," *Networks*, vol. 71, no. 2, pp. 97–106, 2018.

[85] M. T. Beck and J. F. Botero, "Scalable and coordinated allocation of service function chains," *Computer Communications*, vol. 102, no. Supplement C, pp. 78 – 88, 2017.

[86] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*. IEEE, 2014, pp. 7–13.

[87] S. Ghiasvand, F. M. Ciorba, R. Tschüter, and W. E. Nagel, "Lessons learned from spatial and temporal correlation of node failures in high performance computers," in *Parallel, Distributed, and Network-Based Processing (PDP), 2016 24th Euromicro International Conference on*. IEEE, 2016, pp. 377–381.

[88] R. Li, Y. Hu, and P. P. Lee, "Enabling efficient and reliable transition from replication to erasure coding for clustered file systems," *IEEE Transactions on Parallel and Distributed Systems*, 2017.

[89] F. Chen, J.-G. Schneider, Y. Yang, J. Grundy, and Q. He, "An energy consumption model and analysis tool for cloud computing environments," in *Proceedings of the First International Workshop on Green and Sustainable Software*. IEEE Press, 2012, pp. 45–50.

[90] J. Xu and J. a. B. Fortes, "Multi-objective Virtual Machine Placementin Virtualized Data Center Environments," in *2010 IEEE/ACM Int'l Conference on \& Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, ser. GREENCOM-CPSCOM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 179—-188.

[91] Y. Ren, J. Suzuki, A. Vasilakos, S. Omura, and K. Oba, "Cielo: An evolutionary game theoretic framework for virtual machine placement in clouds," in *Proceedings - 2014 International Conference on Future Internet of Things and Cloud, FiCloud 2014*, aug 2014, pp. 1–8.

[92] H. Moens, E. Truyen, S. Walraven, W. Joosen, B. Dhoedt, and F. De Turck, "Cost-Effective feature placement of customizable multi-tenant applications in the cloud," *Journal of Network and Systems Management*, vol. 22, no. 4, pp. 517–558, 2014.

[93] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6463372

[94] H. Moens, B. Hanssens, B. Dhoedt, and F. De Turck, "Hierarchical network-aware placement of service oriented applications in clouds," in *IEEE/IFIP NOMS 2014 - IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World*, 2014, pp. 1–8.

[95] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *Proceedings - IEEE INFOCOM*, apr 2006, pp. 1–12.

[96] M. Ajtai, N. Alon, J. Bruck, R. Cypher, C. Ho, M. Naor, and E. Szemeredi, "Fault tolerant graphs, perfect hash functions and disjoint paths," in *Proceedings., 33rd Annual Symposium on Foundations of Computer Science*, oct 1992, pp. 693–702. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=267781

[97] M. Mihailescu, S. Sharify, and C. Amza, "Optimized application placement for network congestion and failure resiliency in clouds," in *2015 IEEE 4th International Conference on Cloud Networking, CloudNet 2015*, oct 2015, pp. 7–13.

[98] M. J. Csorba, H. Meling, and P. E. Heegaard, "Ant system for service deployment in private and public clouds," in *Proceeding of the 2nd workshop on Bio-inspired algorithms for distributed systems - BADS '10*. ACM, 2010, p. 19. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1809018.1809024

[99] W.-L. Yeow, C. Westphal, and U. Kozat, "Designing and embedding reliable virtual infrastructures," *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures - VISA '10*, vol. 41, no. 2, p. 33, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1851399.1851406

[100] K. B. Laskey and K. Laskey, "Service oriented architecture," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 1, no. 1, pp. 101–105, 2009. [Online]. Available: http://dx.doi.org/10.1002/wics.8

[101] P. D. Justesen, *Multi-objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, 2009, vol. Confirmati.

[102] J. F. Gonçalves and M. G. C. Resende, "Biased random-key genetic algorithms for combinatorial optimization," *Journal of Heuristics*, vol. 17, no. 5, pp. 487–525, 2011. [Online]. Available: http://link.springer.com/10.1007/s10732-010-9143-1

[103] J. Dean and S. Ghemawat, "Simplified data processing on large clusters," *Sixth Symp. Oper. Syst. Des. Implement.*, vol. 51, no. 1, pp. 107–113, 2004.

[104] L. LIU and M. T. ÖZSU, Eds., *Multi-Tier Architecture*. Boston, MA: Springer US, 2009, pp. 1862–1865. [Online]. Available: http://www.springerlink.com/index/10.1007/978-0-387-39940-9{_}652

[105] A. Broder, "Generating random spanning trees," in *30th Annual Symposium on Foundations of Computer Science*, oct 1989, pp. 442–447. [Online]. Available: http://www.computer.org/portal/web/csdl/doi/10.1109/SFCS.1989.63516

[106] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proceedings of IEEE INFOCOM '96. Conference on Computer Communications*, ser. INFOCOM'96, vol. 2. Washington, DC, USA: IEEE Computer Society, 1996, pp. 594–602. [Online]. Available: http://dl.acm.org/citation.cfm?id=1895868.1895900

[107] "Amazon EC2 Instance Comparison," https://aws.amazon.com/ec2/ instance-types, 2016. [Online]. Available: http://www.ec2instances.info

[108] "Nfv, network functions virtualisation and use-cases," ETSI GS NFV 001 V1. 1.1 (2013-10), Tech. Rep., 2013.

[109] Binz, Tobias and Breitenbücher, Uwe and Kopp, Oliver and Leymann, Frank, "TOSCA: portable automated deployment and manage. of cloud applications," in *Advanced Web Services*. Springer, 2014, pp. 527–549.

[110] A. Schrijver, "*Theory of linear and integer programming*", ser. Interscience in discrete mathematics and optimization. Wiley, New York, 1986.

[111] M. R. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman and Company, 1979.

[112] P. Slavík, "A tight analysis of the greedy algorithm for set cover," in *Proc. of the twenty-eighth annual ACM symposium on Theory of computing*. ACM, 1996, pp. 435–441.

[113] E. Q. V. Martins and M. M. B. Pascoal, "A new implementation of yen's ranking loopless paths algorithm," *Quarterly J. of the Belgian, French and Italian Operations Research Societies*, vol. 1, no. 2, Jun 2003.

[114] I. Gurobi Optimization, "Gurobi optimizer reference manual," 2016. [Online]. Available: http://www.gurobi.com

[115] L. Shengquan, W. Chunming, Z. Min, and J. Ming, "An efficient virtual network embedding algorithm with delay constraints," in *2013 16th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, June 2013, pp. 1–6.

[116] "Prometheus - monitoring system & time series database," https://prometheus.io/, 2019, (Accessed on 01/10/2019).

[117] "Build the future of open infrastructure." https://www.openstack.org/, 2019, (Accessed on 01/10/2019).