Last-Backlogged First-Served Deficit Round Robin (LBFS-DRR) Packet Scheduling Algorithm

Dessislava Nikolova and Chris Blondia
University of Antwerp
Dept. Math. and Computer Science
Performance Analysis of Telecommunication Systems Research Group
Middelheimlaan, 1, B-2020 Antwerp - Belgium
and
Interdisciplinary Institute for BroadBand Technology (IBBT)
E-mail: {dessislava.nikolova, chris.blondia}@ua.ac.be

***Abstract*—In this article we present a novel packet scheduling algorithm LBFS-DRR, which combines features from the Last-Come First-Served scheduling discipline and the Deficit Round Robin (DRR) algorithm. In comparison with DRR it provides lower average packet delay, while preserving the advantageous feature like O(1) complexity, fairness, bandwidth guarantee. The lower mean delay is realized by giving service in a round first to flows transmitting bellow their (weighted) fair share. The algorithm exploits the high variability in the typical user traffic pattern resulting in lower mean file transfer delay.**

I. Introduction

In packet communication networks where flows contend for the communication resources one of the most important design issue is the choice of the scheduling algorithm at the queueing points. A wealth of scheduling disciplines are available in the literature and the most simple one is the First-Come-First Served. In high-speed packet routers the algorithms are still mainly based on FCFS having only one queue where the packets from all the flows are queued. Thus the order of arrival of the packets determines the order in which they are forwarded on the output link. However in order to guarantee QoS the network nodes should support per flow queues. Algorithms for per-flow queueing are round robin (RR) and sorted priority algorithms. The Generalized Processor Sharing (GPS) scheduling discipline has emerged as a natural generalization of modeling these algorithms. GPS-based scheduling disciplines are proven to have two desirable properties: (a) they can provide an end-to-end delay bound to a flow whose traffic is leaky-bucket constrained; (b) they can ensure fair allocation of bandwidth among all backlogged sessions regardless of weather or not their traffic is constrained. The sorted priority algorithms approximate very closely GPS but have complexity depending on the number of flows. An example of such algorithms is the WF^2Q+ [1] which is proven to approximate closest the GPS but has $O(logN)$ complexity.

Schedulers based on GPS and having O(1) complexity are the round robin schedulers and from which the most widely studied and deployed in routers is the DRR [2]. The major drawback of DRR is the high latency which depends on the round length. A wealth of schedulers based on DRR which attempt to approximate more closely the ideal GPS and to reduce the latency has been proposed in the literature [3], [4], [5], [6], [7]. They either try to reduce the round duration [7] or to order the packets/flows transmissions to be in closer correspondence to their flows reserved rates which results in increased complexity.

In [8] and [9] the authors proposed an algorithm Priority DRR, which combines FCFS queuing with DRR. They showed that it is scalable and it provides very low average latency for flows transmitting at rates lower than their estimated fair share of the link rate. They proposed an implementation which can have O(1) complexity when the quantum is one packet. Otherwise the complexity depends on setting to zero an array of words with O(N) size, where N is the total number of flows.

In this article we propose a scheduler which combines the DRR and Last Come-First Served (LCFS) scheduling. The stability and asymptotic of LCFS scheduling in its preemptive (PR) and non-preemptive version has been extensively theoretically studied. It has been pointed out that unlike FCFS the LCFS-PR as tasks scheduling has the same fairness as processor sharing and they both are one of the very few "always fair" disciplines [10]. The FCFS policy can be directly applied to packet scheduling by replacing jobs by packets. This is however not true for the LCFS. In a flow packets can not be reordered thus LCFS is not directly applicable for scheduling in packet telecommunication networks. We can however suitably modify it to serve first the last flow which becomes backlogged resulting in Last Backlogged - First Served (LBFS) discipline. We show that the proposed LBFS-DRR algorithm provides low mean transfer delay, guaranteed rate, isolation of the flows and has O(1) complexity.

The rest of the paper is organized as follows. In the next section we give succinct description of the DRR scheduling algorithm. In the the third one we describe the proposed LBFS-DRR algorithm and in the section following we present some results. In the last section conclusions are

drawn.

II. Deficit Round Robin (DRR) scheduling algorithm

In this section we outline the DRR algorithm, a detailed description of which is presented in [2]. Consider number of flows contending for a link and each flow i has a corresponding queue i where the packets belonging to this flow are stored. To each flow is assigned weight w_i and a quantum Q_i proportional to that weight. The quantum indicates the portion of the resources in a round robin cycle a flow should get. If the minimum quantum, say Q_{min} is chosen not less than the maximum packet length in the network then the algorithm has $O(1)$ complexity. The quantums are obtained from $Q_i = w_i Q_{min}$. To each flow i is associated a counter called deficit counter DC_i, which indicates the amount of service the flow can still receive in a round. The flows are serviced in a round robin order. Each round a flow is visited once. Upon a visit to a flow its deficit counter is increased with its quantum. After a packet is sent it is decreased with the size of the packet. In this way if a part of the deficit counter remained unused in a round it will be still available for the next one. Only backlogged flows are serviced. To realise this the DRR scheduler maintains a list of all backlogged flows,refered to in this paper as the $BackloggedFlows$ list. When a flow is no longer backlogged it is removed from the list and its deficit counter is set to 0. When a flow becomes backlogged it is inserted at the tail of the list.

III. LBFS-DRR

Inherent drawback of the DRR is the high average and maximum packet delay and correspondingly file transfer delays. We have identified as one of the reasons for this the fact that when a flow becomes backlogged it is inserted at the tail of the $BackloggedFlows$ list. In this way the flow has to wait for its first service in a backlogged period until all other flows are served their deficit counters. When there are many backlogged flows a new flow can be delayed with a round. As a result even if a flows transmits packets at a rate less than its fair share the delay depends on all backlogged flows.

The most suitable scheduling discipline to address this drawback and which doesn't require sorting of flows is LCFS. However the packet stream in a flow can not be reordered thus LCFS is not directly applicable for scheduling in packet communication networks. We can however modify the LCFS policy such that it serves first the last flow which becomes backlogged resulting in LBFS (Last Backlogged - First Served) discipline.

The LBFS-DRR selects for service the last flow to become backlogged in a round and serves it until either another flow becomes backlogged or it has used up its quantum. In a round all flows receive service proportional to their weight thus assuring that each backlogged flow receives its fair share of the bandwidth.

TABLE I
LBFS-DRR VARIABLES

2 $BackloggedFlows$ lists - lists with backlogged flows.
RC- rounds counter
Per flow:
$Queue_i$, Q_i, DC_i - DRR variables ;
LRS_i - the last round the flow received service

Bellow we give detailed explanantion of the implementation of LBFS-DRR. In order to track the service received from a flow in a round LBFS-DRR needs to distinguish consecutive rounds. Thus besides the DRR flow variables Q_i and DC_i we keep two $BackloggedFlows$ lists. One for the flows which should receive service in the current round as in DRR. The other one for the flows which already have used up their deficit counter for the current round and are still backlogged. The variables for the algorithm are summarized in table I. When the current round $BackloggedFlows$ list becomes empty the two list are switched. Note that this is O(1) operation. The algorithm keeps a global round counter RC which counts the rounds passed thus giving a sort of identification to a round. Each flow stores this identification in a variable Last Round Served LRS_i.

The algorithm can be divided in two processes engaged for the enqueuing of packet and dequeuing of a packet.

A. Enqueue

The pseudo code of the enqueue process is given in table II.

When a flow becomes backlogged it is examined whether it has already received service in the current round by comparing its LRS_i variable with the RC. In case it is so the enqueue process continues further to check whether the flows deficit counter is sufficient for the packet. If not the flow is inserted in the next active list to receive service in the next round. This is important check as it insures that no flows with insufficient counter are found in the list which is

TABLE II
ENQUEUE PROCESS

```
1. i=p.Flow();
2. Queue_i.Insert(p);
3. IF(i not in any list)
4.    if(LRS_i == RC)
   /*The flow has already received service in this round.*/
5.       IF(Length(p) ≤ DC_i)
6.          CurrentActiveList.push-front(i);
7.       else
8.          DC_i += Q_i;
9.          NextActiveList.push-back(i);
10.   else
   /*The flow hasn't received any service in this round.*/
11.      DC_i = Q_i;
12.      CurrentActiveList.push-front(i)
```

TABLE III
DEQUEUE PROCESS

```
13. WHILE (Any Active list NOT empty)
14.   IF(CurrentActiveList NOT empty)
15.         flow-iterator = Begin(CurrentActiveList);
16.         i= flow-iterator.flow;
17.   else
18.         swap(ActiveList1 and ActiveList2);
19.         RC++;
20.         break;
21.   p=Head(Queue_i);
22.   send(p);
23.   DC_i = DC_i − p.Length();
24.   IF(Queue_i.empty()
25.     LRS_i = RC
26.     CurrentActiveList.erase(flow-iterator);
27.   else /*the queue is not empty*/
28.     IF(p.Length() DC_i) the DC is over. */
29.       DC_i+ = Q_i;
30.       CurrentActiveList.erase(flow-iterator);
31.       NextActiveList.push-back(i)
```

necessary in order to achieve O(1) complexity. If the flow is found to have sufficient deficit for the current round it is inserted it at the front of the active list. This ensures the LBFS part of the algorithm.

B. Dequeue

The pseudo code of the dequeue process is given in table III. The dequeue process selects the head of the current active list for service. After a packet is transmitted and the DC of the flow updated the scheduler has to check whether the flow is still eligible for transmission in the current round. This includes that the flow is backlogged and the DC is sufficient for the next packet in the flows queue to be transmitted. Otherwise the flow is erased from the current active list. When the scheduler selects a flow for service it doesn't pop it from the list but rather keeps a pointer to the element in the list. Thus after a packet transmission even if other flows have been inserted in the list the flow can be still removed from it in O(1) operation. For the reader without programing knowledge list is a data structure in which each element besides the data itself keeps pointer i.e. the address in the memory where the next element is kept. Thus an element from the start, the end or on any other position is removed or inserted in constant time.

C. Discussion

In this implementation we have chosen to keep track of the different rounds by having two lists. There are other possibilities like for example the one realized in [7]. There a counter of the elements in the list is kept. However we find that this solution is less suitable for LBFS-DRR as the number of flows is highly variable during one round.

We use as the RR scheduling the DRR algorithm but it can be replaced with some other RR scheduling algorithm like for example SRR [11]. It is maybe the most close to DRR algorithm and as it was proven in [12] has lower latency than DRR. The importance of the latency will be explain in more detail in section IV-B.

The algorithm can be easily modified in a simpler non-quantum-preemptive version where when a flow is chosen for service it is popped from the list and served until it has used its quantum or is no longer backlogged. In other words no flow can preempt the service of flow in a round until the later hasn't exhausted its DC.

IV. Performance Evaluation

A. Complexity

The complexity of the proposed algorithm is O(1). As it is seen from the pseudo code all the operations used are performed in constant time. Further on after a packet is sent the scheduler checks the eligibility for another packet to be sent from the flow. When the flow is not eligible it is cleared from the current active list. When a flow is to be added to the active list its eligibility is also checked so no ineligible flows are inserted in the current active list.

B. Latency

A technique to compare different schedulers can be found in the theory of Latency-Rate servers [13]. Within the theory a scheduler is characterized with a certain latency which is obtained from the lower bound on the service W_i offered to the packets of flows i by the scheduler and defined as

$$W_i(t_0, t) \geq \rho_i(t - t_0 - \theta_i),$$

where θ_i is the latency, r_i is the reserved rate of flow i, t_0 is a start of a busy period, t is whatever time within this busy period and busy period is the maximum period in which the arrival rate is more or equal to the guaranteed rate for flow i and at t_0^- the flow wasn't backlogged. Further on it can be shown that for a source with arrival rate which is leaky-bucket constrained with parameters (ρ_i, σ_i) a bound on the maximum delay experienced by a packet is given by

$$D_{max} \leq \frac{\sigma_i}{\rho_i} + \theta_i.$$

Following the derivation for the latency of the DRR algorithm [14] it is straightforward to prove that the DRR and the LBFS-DRR algorithms have the same latencies. This means that they guarantee the same maximum packet delay for leaky bucket constrained traffic.

C. Fairness

As a quantitative measure of fairness of scheduling algorithms can be used the Golestiani fairness and its modified in [13] version and can be seen as the maximum absolute difference between the normalized service received by two flows

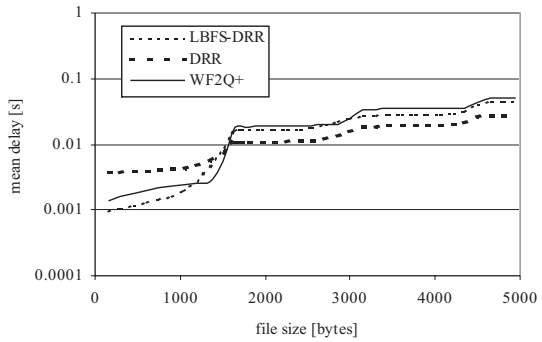


Fig. 1. Mean transfer delay vs. file size

over any time interval in which they are both backlogged.

$$\left|\frac{W_i(\tau,t)}{r_i} - \frac{W_j(\tau,t)}{r_j}\right| \leq \Phi^S,$$

where the interval (τ, t) is such that the two compared connections i,j are continuously backlogged. An algorithm is fair if its fairness measure is bounded i.e Φ^S. This measure has been used in a variety of works [11], [13], [15] to calculate the fairness of a number of algorithms. The fairness of the DRR algorithm is bounded and depends on the round size.The LBFS-DRR can have different from DRR service order in a round. However for backlogged flows this order remains the same. Moreover the amount of service received by a backlogged flow is also the same for the two algorithms. Therefore the two algorithms have the same fairness.

D. Simulation results

We have evaluated the performance of the LBFS-DRR by means of a simulation program implemented in OMNET++ [16]. In all the scenarios we compare the performance of LBFS-DRR with DRR and WF^2Q+ scheduling algorithms. We report two sets of simulations. In the first we demonstrate the reduction of the mean delay for different file size distributions. In the second we show the isolation of the flows and the dependance on the reserved rate for the weighted version. The simulation setup is adopted from the Ethernet based networks like Hybrid Fiber Coax (HFC) and Ethernet Passive Optical Networks, where in the downstream there is one shared link and a number of flows are contending for bandwidth. The maximum packet size is the maximum size of the Ethernet frame - 1518 bytes. In the first simulation scenario the network link rate is R=10 Mbits/s and there are 41 active flows each transmitting at rate 0.95*250Kb/s which results in 0.95*Link rate. The file sizes have pareto bounded distribution with parameters (24, 15800, 1.1). The quantum sizes for all flows for the DRR and LBFS-DRR are 1518 bytes.

On fig. 1 the mean file transfer delay vs. the file size is shown. The delay function is stepwise because the Ethernet layer fragments the file received from the application layer in packets with maximum length of 1518. For files with size bellow the quantum the LBFS-DRR achieves the lowest delay. It is expected as they preempt the service of all other backlogged flows. For all other file sizes bigger than the quantum it is the DRR scheduler which provides them with the lowest delay. Ones queued they receive service in a round before other flows which become backlogged meanwhile.

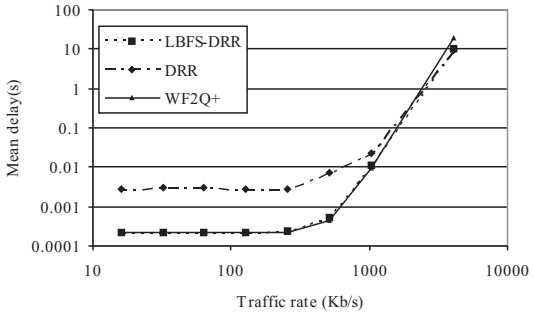
TABLE IV
PACKET DELAY

	Mean (ms)	Max (ms)	StdDev (ms)
DRR	3.78	256.28	4.45
LBFS-DRR	1.31	308.22	4.48
WF2Q+	1.78	343.33	5.11

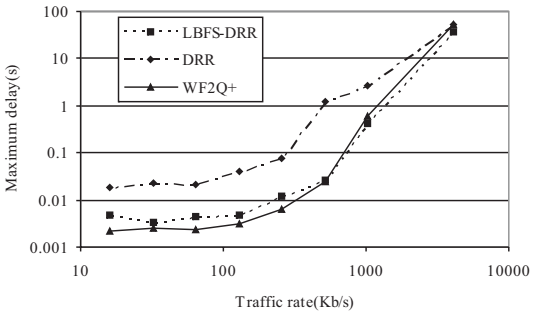
In table IV the results for the average and the maximum packet delay and standard packet delay deviation per flow are given. As it can be seen for heavy tailed distributed file sizes the lowest mean delay is achieved with LBFS-DRR. For such distribution the majority of the files are small. Thus when the user experienced delay is averaged out over all files the small ones deliver the highest contribution. It is founded to use heavy tailed distributions for file sizes as they are close to typical user traffic pattern as shown in [17].

In the second simulation scenario we study the packet level performance of the scheduler i.e. the packet queueing delay. We want to demonstrate that flows which transmit bellow their respective share experience lower packet delay than flows transmitting more than that. We simulate link with rate R=40Mb/s and N=160 flows contending for the bandwidth. The flows are divided in 8 groups depending on the rate they generate traffic. The lowest is 16 Kb/s and the highest 4Mb/s. The packet sizes are drawn from distribution reported in [18] and the generation process is ON-OFF with pareto distributed ON times and exponentially distributed OFF times and packet inter-arrival times.

On figures 2(a) and (b) are given the respectively the average and maximum delay and each point is obtained from the delays of all the packets from all the flows in a group. The fair share of a flow if all are backlogged is $R/N = 250Kb/s$. As it can be seen flows which generate at rate bellow or at their fair share experience lower delay. Packets from flows with lower rate than their fair share on arrival at the scheduler will find their corresponding queue empty and they are likely to be inserted at the head of the current active list. Their delay will not be influenced from the backlogged flows but only from new arrivals from the other flows. From the figures it can be seen that the behavior of the LBFS-DRR algorithm is closer to the one of the WF^2Q+. The delay for flows transmitting bellow their fair share under the DRR scheduling discipline is found to be ≈ 10 times worse than the one experienced with the other two disciplines. Even though the delay for the flows transmitting at lower rate is significantly lower for LBFS-DRR this doesn't result in significantly higher delay for the



(a)



(b)

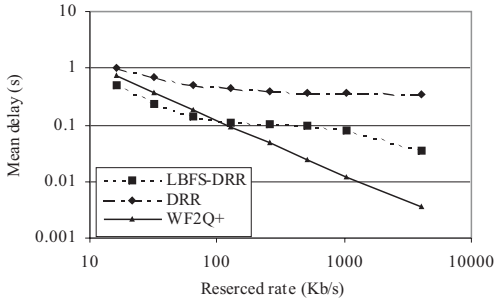
Fig. 2. Packet queueing delay with all $w_i = 1$ traffic load $\approx$ Link Rate (a) average delay and (b) maximum delay.

flows transmitting at higher rate.

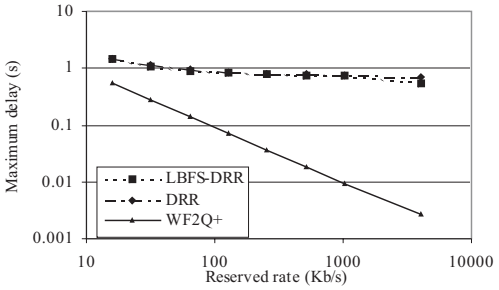
In the third simulation scenario we aim to demonstrated the performance of the weighted version of the LBFS-DRR. The 160 flows are divided in groups with different weights ranging from 1 to 256. The sum of all weights is 2500 and weight $w_i = 1$ corresponds to $r_i = 16$ Kb/s reserved rate. Each point on the figures is obtained by averaging or taking the maximum value of the packet delays from all the packets from all flows from the same group. The total guaranteed rate is the same as the link capacity i.e. $\sum_i r_i = R$. Part of the flows - "well-behaved" - are transmitting traffic at their guaranteed rate with leaky bucket parameters (r_i, 1518). Another part "misbehaiving" comprising of 32 flows are transmitting above their respective reserved rates and are expected to be constantly backlogged. On figure 3(a) is given the average delay of the "well-behaved" flows versus their reserved rate.

The packets from well behaved flows precede the packets from the 32 constantly backlogged ones the later don't influence the mean delay of the former. As a result the mean delay experienced by the packets under the LBFS-DRR scheduling is less than under DRR.

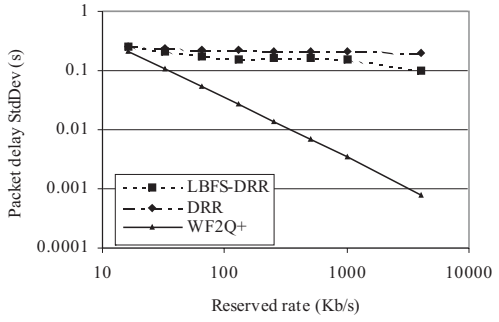
The WF^2Q+ scheduler serves the flows at their relative reserved rate thus it can delay packets from flows with lower reserved rate in the presence of backlogged flows with high reserved one. The LBFS-DRR scheduler as a sort of



(a)



(b)



(c)

Fig. 3. (a) The mean packet queueing delay vs. the reserved rate of its corresponding flow. (b) The maximum packet queueing delay vs. the reserved rate of its corresponding flow. (c) The standard deviation of the packet delay vs. the reserved rate of its corresponding flow .

a round robin algorithm tends to equalize the delay of the flows in a round regardless of their reserved rate. When a packet from a non-backlogged flow say i arrives at the LBFS-DRR scheduler at time t it is inserted at the front of the current active list. Only packets from non-backlogged flows which arrive after this time instant t influence the packet delay of flow i. When a packet from a flow with low reserved rate arrives at a LBFS-DRR scheduler there is a certain probability that it will precede packets from flows with higher reserved rates. While waiting for service new flows transmitting at higher rate will become backlogged but still in comparison with WF^2Q+ its delay will be lower. For the flows with higher rates the opposite scenario happens.

With bigger probability they will precede packets from flows with lower rates. However while waiting for service, packets from flows with lower rate might arrive which will result in higher delay than the one experienced under the WF^2Q+ discipline

The maximum delay as pointed out in section IV-B is the same for LBFS-DRR and DRR as presented on fig 3(b). To complete the analysis of this scenario on fig 3(c) we show the standard deviation of the packet delay. For LBFS-DRR it is in the order of DRR but of course expressed in percentage of the mean the LBFS-DRR has the highest packet delay standard deviation and correspondingly variation.

V. Conclusions

In the paper we have presented a novel scheduling algorithm which combines the DRR with the LBFS discipline. With LBFS-DRR we have designed a scheduler which combines all the desirable properties of the DRR and LBFS algorithms and discards their drawbacks. When all flows are behaving well due to the variable file size, which is typical for the user traffic the LBFS-DRR has lower mean transfer delay. In the presence of flows transmitting more than their momentous fair share, the well behaved flows are not affected, they remain isolated and even stronger their packet delay under the LBFS-DRR is close to the delay of a sorted priority scheduling algorithm. The LBFS-DRR is a packet scheduling algorithm with O(1) complexity and bounded delay.

References

[1] J. Bennett and H. Zhang, "Hierarchical packet fair queueing algorithms," *IEEE/ACM Transactions on networking*, vol. 5, no. 5, October 1997.

[2] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round robin," in *Proc. SIGCOMM'95*, 1995.

[3] S. Ramabhadran and J. Pasquale, "The stratified round robin scheduler: design, analysis and implementation." *IEEE/ACM Trans. Netw.*, vol. 16, no. 6, pp. 1362–1373, 2006.

[4] L. Lenzini, E. Mingozzi, and G. Stea, "Eligibility-based round robin for fair and efficient packet scheduling in wormhole switching networks." *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 3, 2004.

[5] G. Chuanxiong, "Srr: An o(1) time complexity packet scheduler for flows in multi-service packet networks'," in *Proc. SIGCOMM'01*, 2001.

[6] S.-C. Tsao and Y.-D. J. Lin, "Pre-order deficit round robin: a new scheduling algorithm for packet-switched networks." *Computer Networks*, vol. 35, no. 2-3, pp. 287–305, 2001.

[7] S. S. Kanhere, H. Sethu, and A. B. Parekh, "Fair and efficient packet scheduling using elastic round robin," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 324–336, 2002.

[8] A. Kortebi, S. Oueslat, and J. Roberts, "Implicit service differentiation using deficit round robin," in *Proc. ITC19*, Beijing, 2005.

[9] A. Kortebi, L. Muscariello, S. Oueslat, and J. Roberts, "Minimizing the overhead in implementing flow-aware networking," in *Proc. ANCS'05*, Prinston, New Jersey, USA, October 2005.

[10] A. Wierman, "Fairness and classifications," *ACM SIGMETRICS Performance Evaluation Review*, vol. 34, no. 4, March 2007.

[11] H. Adisshu, G. Parulkar, and G. Varhese, "A relaible and scalable striping protocol," in *Proc. ACM SIGCOMM*, August 1996.

[12] D. Nikolova and C. Blondia, "Evaluation of surplus round robin scheduling algorithm," in *Proc. SPECTS'06*, Calgary, Canada, August 2006.

[13] D. Stiliadis, "Traffic scheduling in packet-switched networks: Analysis, design, and implementation," Ph.D. dissertation, University of California at Santa Cruz, USA, June 1996.

[14] S. Kanhere and H. Sethu, "On the latency bound of deficit round robin'," in *Proc. ICCCN*, Miami, Florida, USA, October 2002.

[15] L. Lenzini, E. Mingozzi, and G. Stea, "Tradeoffs between low complexity, low latency, and fairness with deficit round robin schedulers," *IEEE/ACM Trans. Netw.*, vol. 12, no. 4, August 2004.

[16] Omnet++ simulator. [Online]. Available: www.omnetpp.org

[17] Guo and Mata, "The war between mice and elephants," in *Proc. ICNP 2001*, Riverside, CA, November 2001.

[18] K. Thompson, G. Miller, and R. Wilder, "Wide-area internet traffic patterns and characteristics," *IEEE Network*, vol. 1, no. 6, pp. 10–23, 1997.