# DRR–SFF: A Practical Scheduling Algorithm to Improve the Performance of Short Flows

Changhua Sun, Lei Shi, Chengchen Hu, Bin Liu
Department of Computer Science and Technology, Tsinghua University, China
{sch04, shijim, hucc03}@mails.tsinghua.edu.cn, liub@tsinghua.edu.cn

*Abstract*— **Short flow first scheduling (*SFF*) strategy is effective in obtaining more stringent performance bounds for short flows in Internet. However, previous strict SFF approaches invested the short flows with excessive preference leading to the starvation of other long flows. Moreover, these SFFs are hard to be deployed due to either extreme complexity or the modification of TCP protocol. Inspired by the fairness and practicality of *Deficit Round Robin (DRR)*, we proposed a novel scheduling mechanism, namely *Deficit Round Robin with Short Flow First (DRR-SFF)*, which improves the performance of short flows with limited penalizing long flows. DRR-SFF uses *Weighed DRR* to schedule short and long flows respectively and treats long flows more fairly. Through trace-driven simulation, we show that the mean transmission time and loss rate of short flows under DRR-SFF are significantly reduced compared with FIFO scheduling using DropTail. Meanwhile, the performance of long flows under DRR-SFF does not degrade much, retaining to that of FIFO scheduling. Our results demonstrate DRR-SFF is superior to strict SFF approaches, as the latter drives long flows into starvation in our simulations. Moreover, DRR-SFF inherits the *O(1)* computation complexity of DRR, which makes it easy to be deployed in edge routers.**

## I. Introduction

Researches in [1]–[4] emphasized that *short flows* (In this paper, short flows are defined as flows with flow size less than a predefined threshold and long flows otherwise.) should be given higher priority over the other *long flows*, because 1) short flows are mainly generated by delay–sensitive interactive ser–vices, such as web, telnet, ssh and VoIP; 2) When experiencing packet losses, short TCP flows rely on the *retransmission timeout* to detect loss, while long TCP flows can trigger the duplicate ACK mechanism followed by a *fast retransmission*. This causes short TCP flows to have high variations in transfer times; 3) Internet flow size distributions exhibit heavy tailed characteristic [5], [6] and most flows are short, but most bytes are in long flows [7], [8]. In summary, a scheduling strategy to favor short flows looks to be a must.

However, designing a scheduling algorithm that favors short flows is not straightforward, several issues should be solved: 1) how to distinguish between short and long flows; 2) how to design the scheduling strategy which could be easily deployed; 3) how to favor short flows without starving long flows.

In several related works, RuN2C [3], a threshold based two–queue approach, enqueued the first *th* packets of a flow in the first queue, and the remaining packets in the second queue. Packets from the second queue were not served unless the first queue was empty. Both queues were served in a FIFO disci–pline. In order not to maintain flow states, RuN2C modified TCP sequence number to detect whether the packet was put into the first or second queue. Hence, TCP sequence number should start from a set of possible initial numbers, which reduced the randomness of TCP initial sequence numbers and would lead to security problems such as IP address spoofing and session hijacking [9].

LAS (Least Attained Service) [4], a size based per–flow scheduling approach, would always schedule packets belong–ing to the flow that had received the least amount of service. LAS used a priority assignment unit to compute the service priority of each flow and sort the list in decreasing order of priority. It required $O(\log n)$ computations in scheduling each packet, where $n$ was the number of active flows. In addition, LAS may be extremely unfair for long flows starting during the service time of other long flows with almost the same flow size, because these long flows would be finished almost at the same time in spite of their different starting time.

Ideas of these strict short flow first (SFF) approaches are good but not complete. Their motivation is based on the assumption of burst characteristic of traffic: there are "holes" between the bursts of short flows where long flows can fill in [2]. Normally, this assumption matches the characteristics of real trace. However, some special cases, such as DDoS attack using many short flows (like SYN flooding and DNS amplification attacks [10]), may overthrow the assumption. Moreover, they were unfair among flows, hard to be deployed, and might starve long flows.

Aiming at these issues, a novel scheduling scheme, *Deficit Round Robin with Short Flow First* (*DRR-SFF*), is proposed. DRR–SFF is on the basis of *Deficit Round Robin (DRR)*, a widely used fair packet scheduling strategy [11], [12]. Our main idea is using *Weighed Deficit Round Robin* to schedule short and long flows respectively. Packets with the same *5-tuple* (source and destination IP, source and destination Port, Protocol) are classified as the same flow. Flows are organized as two groups, *Prioritized group (PQ)* and *Best-effort group (BQ)*. Each flow is first put into PQ and then moved to BQ

This work is supported by NSFC (No. 60373007, 60573121 and 60625201), the Cultivation Fund of the Key Scientific and Technical Innovation Project, Ministry of Education of China (No. 705003), the Specialized Research Fund for the Doctoral Program of Higher Education of China (No. 20040003048 and 20060003058), China/Ireland Science and Technology Collaboration Re–search Fund (2006DFA11170), and the Tsinghua Basic Research Foundation (JCpy2005054).

after *th* bytes have been scheduled. As PQ is assigned higher priority than BQ, flows with flow size less than *th* bytes are favored, while long flows in BQ would not be starved since PQ does not have strict priority over BQ.

We describe detailed DRR–SFF scheduler as well as the setting of parameters in the algorithm. Through simulation, we compare DRR–SFF with FIFO scheduling and strict SFF approach LAS [4]. It is shown that under DRR–SFF, as compared with FIFO using DropTail, the mean transmission time and loss rate of short flows are significantly reduced just as those under LAS. Meanwhile, the performance of very long flows under DRR–SFF is close to that under FIFO, and achieves 25% improvement compared with LAS (measured by the mean transmission time). The contribution of this paper lies in three main aspects:

- DRR–SFF is fair among short flows and among long flows by using fair packet scheduling;
- DRR–SFF improves the performance of short flows without penalizing long flows much since it does not give strict priority to short flows and treats long flows fairly;
- DRR–SFF is practical and easy to be deployed in routers, especially edge/access ones.

The rest of the paper is organized as follows. Section II presents the DRR–SFF solution. Section III discusses the implementation issues of DRR–SFF, including flow timeout value, threshold value, the scalability of per–flow scheduling, and bandwidth allocation between short and long flows. Section IV carries out simulations to evaluate the performance of DRR–SFF. Finally in Section V, we conclude the paper.

## II. DRR–SFF

Our motivation of DRR–SFF is as follows: 1) to improve performance of short flows and fairly treat long flows; and 2) to be easily deployed in routers. We first give the main idea of DRR–SFF, and then describe its detailed design.

### A. Main Idea

DRR–SFF is on the basis of *DRR*, a widely deployed fair scheduling scheme, (e.g., in Cisco routers [12]), which can be considered as the round robin scheduling for variable–length packets. Moreover, DRR is especially easy to be implemented by hardware. DRR–SFF inherits the characteristics of DRR and can be easily implemented by hardware or software.

Our classification of short and long flows is as follows: short flows are those with flow size less than a threshold *th* and long flows otherwise [2]. Flow size is the total packets or bytes of the flow. We divide queues into two groups: prioritized queue group (PQ) and best–effort queue group (BQ). The first *th* bytes of a flow are served in PQ and the remaining bytes are served in BQ. Inside the two groups, both of them serve flows using a DRR discipline. Unlike the strict–priority, two–queue–based approaches in [1]–[3], we implement the *Weighed DRR* scheduling between two queue groups. Each group is offered services proportional to its assigned weight. A *deficit counter (DC)* is associated with each group. During each round of scheduling, the *DC* is incremented by a *quantum*, representing the weight of its group. Then this group is served as long as its *DC* is greater than zero. The *DC* is decremented by the amount of bytes served and is carried over to the next round. When a group becomes empty, its *DC* is immediately reset to zero. Through this weighted DRR, the priority of the two groups can be determined by setting their quanta. We introduce the concept of *quantum ratio* to represent our preference between short and long flows, which is defined as the ratio of PQ's quantum to BQ's quantum. In this way, although DRR–SFF will always assign higher priority to short flows by offering them larger speedup, the long flows would still hold a fraction of service, thus would not be starved. Particularly, in some special cases, short flows may contribute more bytes than normal condition, and strict SFF approaches may degrade the performance of long flows too much while DRR–SFF still treats them fairly. Meanwhile, the mean transmission time of short flows in our approach will only slightly increase compared with LAS or RuN2C.

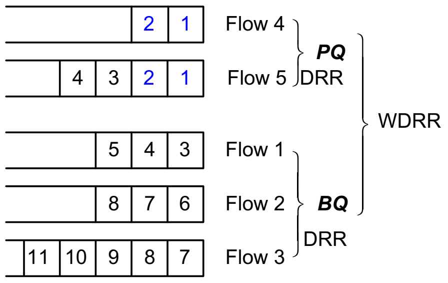

Fig. 1. Illustration of DRR–SFF

### B. Architecture and Algorithm Design

We first show how DRR–SFF works by an example. Fig. 1 illustrates DRR–SFF using two packets as the threshold. Suppose there are five flows in the external buffer. Since the short parts of *Flow 1, 2* and *3* were served in PQ, they are now served in BQ. *Flow 4* and *5* are new flows, so they are in PQ. After their first two packets are served, *Flow 4* is empty and would be removed, while *Flow 5* would be moved to BQ.

To implement DRR–SFF in routers, two *active flow lists (AFLs)*, storing data for active flows, are kept for PQ and BQ. The data in *AFL* include *flow identity*, *DC* of the flow, *pkts*(to track how many packets of the flow in buffer) and *bytes_scheduled*(to track how many bytes the flow has transmitted in PQ). A flow is active when it has packets in external buffer or we could see packets of the flow for every certain interval. The *AFLs* are organized as bidirectional circular linked lists. In each scheduling round, the head entry of the *AFL* is visited and head pointer may be updated by the next entry of the original head entry. A flow is put in the tail entry of the list when inserted in the *AFL*. These two *AFLs* are made up of continuous memory for quick visit and update. Besides two *AFLs*, a *flow table*, indexed by *flow identity*, keeps the pointers of the first and last packet in the external buffer. This table is employed to enqueue and dequeue packet into

```
Initialization:
high_low_turn = 1;
Enqueueing module: on arrival of packet p
if no free buffers left then
 FreeBuffer(); //using buffer stealing
 i = ExtractFlow(p);
 i->pkts ++;
 if(i->pkts == 1) //new flow
  i->DC = quantum;
  if (i->bytes_scheduled < threshold)
  //active in PQ group
   InsertActivelist_PQ(i);
  else// active in BQ group
   InsertActvielist_BQ(i);
Enqueue(i,p); //enqueue packet p to queue i
```

Fig. 2. Enqueueing module of DRR-SFF

```
Dequeueing module:
Either PQ or BQ is null, it is other group's turn;
if(high_low_turn AND Activelist_PQ not null)
 if(high_DC <= 0) high_DC += quantum_high_;
 Remove the head of Activelist_PQ, say flow i
 if(i->DC <= 0) i->DC += quantum;
 p=Dequeue(i);
 Packetsize = Length(p);
 high_DC -= Packetsize;
 i->DC -= Packetsize;
 if(high_DC <= 0) high_low_turn = 0;
 i->bytes_scheduled += Packetsize;
 if(i->bytes_scheduled >= threshold) {
  if(empty i) remove i from Activelist_PQ
  else move i to Activelist_BQ
 }else if(i->DC <=0 ) move i to Activelist_PQ tail
else if(!high_low_turn AND Activelist_BQ not null)
 if(low_DC <= 0) low_DC+=quantum_low_;
 Remove the head of Activelist_BQ, say flow i
 if(i->DC <= 0) i->DC += quantum;
 p=Dequeue(i);
 Packetsize = Length(p);
 low_DC -= Packetsize;
 i->DC -= Packetsize;
 if(low_DC <= 0) high_low_turn = 1;
 if(empty i) remove i from Activelist_BQ
 if(i->DC <=0 ) move i to Activelist_BQ tail
```

Fig. 3. Denqueueing module of DRR-SFF

and out of buffer. Fig. 2 and Fig. 3 present the pseudo-codes. Notably, the pseudo-codes do not consider flow timeout, which is discussed in III-A.

***Enqueue Operation*** If on the arrival of a packet *p* the buffer is full, a selected packet will be dropped using buffer stealing [11], [13] (first drop packet from BQ, then PQ). If the flow *i*, extracted from the 5-tuple of packet *p*, has one packet and its *bytes_scheduled* is less than threshold *th*, flow *i* is inserted at the tail of Activelist_PQ. If flow *i* has one packet but its *bytes_scheduled* is not less than *th*, it is inserted at the tail of Activelist_BQ. Otherwise, flow *i* is already in the active lists.

***Dequeue Operation*** When it is PQ's turn, if its *deficit counter(DC)* is not more than zero, its *DC* is added by its *quantum*. In PQ, flows are scheduled using DRR. The first packet *p* of the head entry, flow *i*, in the PQ active list is dequeued and the bytes of packet *p* is subtracted from flow *i*'s *DC*. If flow *i*'s *DC* is not more than zero, the head entry is changed to the next entry of flow *i* in PQ active list. At the same time, the bytes of packet *p* are also subtracted from PQ's *DC*. If PQ's *DC* is not more than zero, the next turn is BQ's. Otherwise, we will still schedule in PQ. When scheduling in PQ, if a flow's *bytes_scheduled* is not less than *th*, and it is not empty, the flow is moved to BQ. Scheduling in BQ is the same as that in PQ. Particularly, we may schedule in BQ while PQ is not empty. This is the main difference between our approach and other threshold-based approaches.

In addition, DRR-SFF does not disturb the packet order inside each flow since the long flows are moved to BQ after their short parts have been scheduled in PQ.

## III. Implementation Issues

To further understand the algorithm of DRR-SFF, we discuss some important implementation issues and analyze its complexity in this section.

### A. Flow timeout

Many short flows are carrying interactive traffic. They may be silent after a while and continue to send data. To improve their performance and reduce the flow state information, we introduce flow timeout. If we do not see any packets with the same 5-tuple for a certain period, the flow is timeout and we can clear the flow state information. Through flow timeout, interactive flows may be always treated as short flows and put in PQ, and flow state information is decreased, which would reduce the complexity of keeping per-flow state. To implement flow timeout in DRR-SFF, we need to keep time stamp for active flows. The time stamp is the arrival time of last packet in the flow. Then it is checked whether flows are timeout periodically or on scheduling. The value of flow timeout relies on the traffic and system capacity to hold information of active flows. Research in [14] shows 4 to 64 seconds are suitable for flow timeout.

### B. Threshold value th

As for the threshold *th*, 20 packets or 20K bytes is suitable. This threshold is also discussed in [2], [3]. According to TCP protocol, about 8 packets should be given priority until the congestion window reaches a value of 3 or 4 to avoid timeout. So the threshold larger than 8 is feasible. On the other hand, the threshold is to distinguish short and long flows and the measurement results are also important. It is better to adjust threshold th to fit the change of the characteristics of traffic. If short flows with some threshold contribute more bytes than usual, it is difficult to favor short flows without penalizing long flows using strict SFF. As the fairness property of DRR-SFF, if we choose some appropriate *quantum ratio* (discussed in Section III-D), DRR-SFF can still improve the performance of short flows while treating long flows fairly (illustrated in Section IV-C). The threshold, 20K bytes, is introduced by our real trace analysis to improve most delay–sensitive interactive traffic.

### C. *Per-flow scheduling*

DRR–SFF is easy to implement when keeping per–flow state. Per–flow scheduling is not popular in research area because of scalability concern. But we should notice that our approach, DRR–SFF, is mainly used in edge routers for the consideration that most congestion currently in Internet occur at the edges, and also the number of active flows is moderate. Recent measurement in [6] shows that the average number of active flows per minute is less than 50,000 for OC–12 links and less than 300,000 for OC–48 links, and per–flow scheduling may be feasible in hardware on access links. Moreover, state–of–art Network Processor, like AMCC nPX5700 chip set, can support per–flow queueing for 512K flows easily [15]. Therefore, it is not impossible to deploy per–flow scheduling in access routers.

### D. *Analysis on quantum ratio*

Suppose bytes of the short flows contribute $\theta$ and *quantum ratio* is $\alpha$. From the property of DRR–SFF's round robin, we can roughly draw this conclusion: DRR–SFF schedules $\alpha$ packets of short flows when scheduling one packet of long flows. To obtain equal service between short and long flows, we should first satisfy (1). Then we get (2).

$$\alpha/(\alpha+1) > \theta. \tag{1}$$
$$\alpha > \theta/(1-\theta). \tag{2}$$

For example, if short flows contribute 20% bytes, then $\alpha > 0.25$ can ensure fairness of short flows towards long flows. On the other hand, if $\alpha < 1$, then DRR–SFF may schedule one packet of short flow when scheduling several $(1/\alpha)$ packets of long flows. The priority of short flows is not enough to improve their performance when multiple packets of short and long flows exist. So $\alpha > 1$ is recommended.

As the increasing of $\alpha$, DRR–SFF is approaching strict SFF. From our simulation, *quantum ratio* around 2 is appropriate for most conditions, as performance of short flows is improved while fairness of long flows is guaranteed.

### E. *Complexity of DRR-SFF*

The complexity of DRR–SFF mainly depends on the per–flow scheduling, which is also required in DRR. DRR–SFF is almost the same as DRR except the addition *active flow list* and more information in the *flow table*. These just add a little more memory. When scheduling one packet, time complexity of DRR–SFF is *O(1)*, which is the same as that of DRR. Therefore, DRR–SFF inherits the characteristics of DRR and can be easily implemented by hardware.

## IV. Performance Evaluation

### A. *Simulation Methodology*

We carry out simulations with ns–2 [16] to evaluate the performance of DRR–SFF. Fig. 4 shows the network topology applied in our simulations, the same as that in [2], [4]: Totally

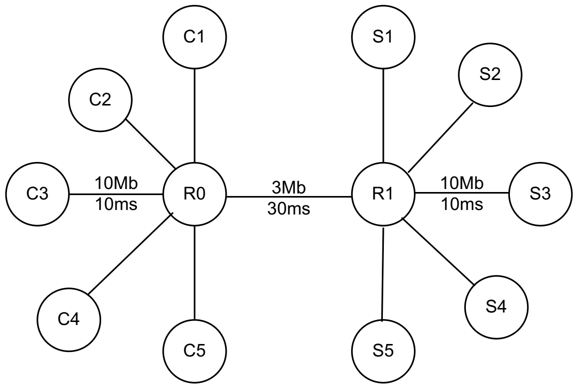

Fig. 4. Network topology in simulations

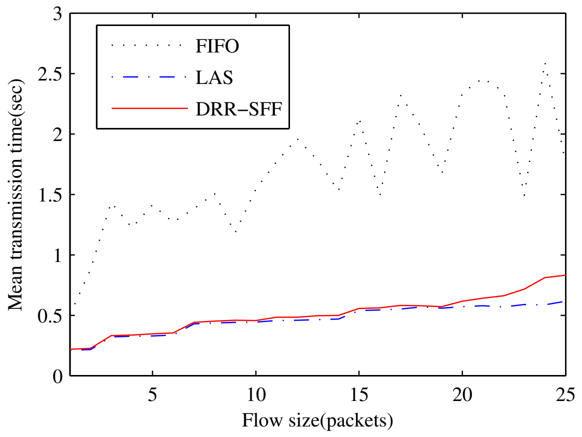

Fig. 5. The mean transmission time of short flows

5 client hosts (C1–C5) and 5 servers (S1–S5) are connected to router R0 and R1 separately by 10 Mbps links with 10 ms propagation delay; R0 and R1 are connected by a bottleneck link with 3 Mbps bandwidth and 30 ms propagation delay.

We randomly choose a server from S1–S5 as TCP source and send flows to a randomly chosen client from C1–C2. The total flow number in one simulation is 4000. And the flow size follows the well–known Pareto distribution [5] with shape parameter 0.9252, minimum 1 packet (scale 1). The parameters of flow size distribution are obtained according to the traffic trace at the egress point of Tsinghua University [17]. We adjust the arrival rates to obtain a load rate of 0.8. The size threshold for long flows is set to 20K bytes, the *quantum ratio* of DRR–SFF is set to 2, and the *flow timeout* is set to 10 seconds. Using these parameters, more than 90% of the total traffic flows in our simulation are short and they only contribute about 20% of the total network load.

### B. *Simulation Results*

We mainly compare LAS and DRR–SFF as only this two adopt the per–flow scheduling scheme, furthermore, LAS is one of the typical strict short flow first approaches (SFF). The

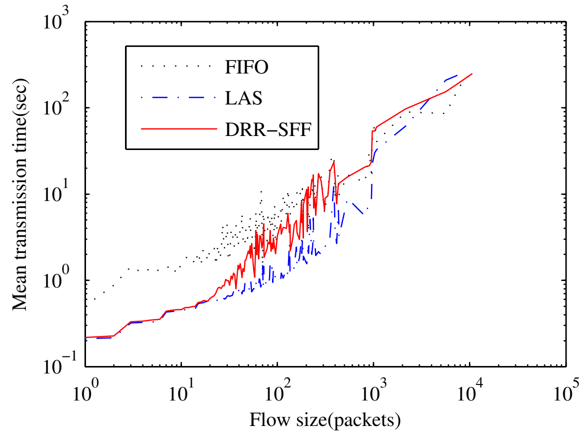

Fig. 6. The mean transmission time of all flows

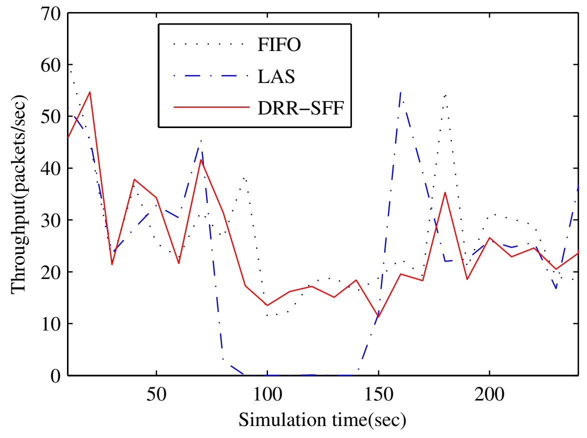

Fig. 8. Throughput of a single long FTP flow

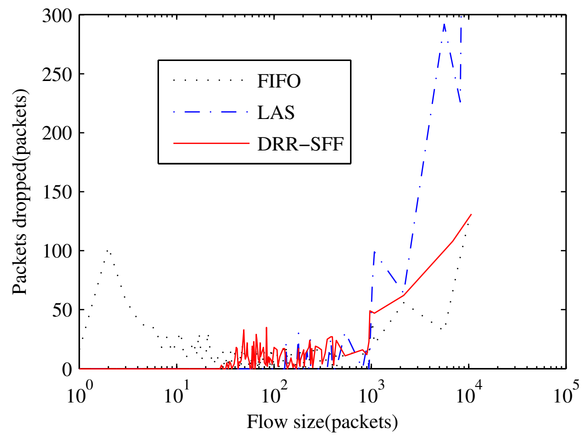

Fig. 7. The number of dropped packets of all flows

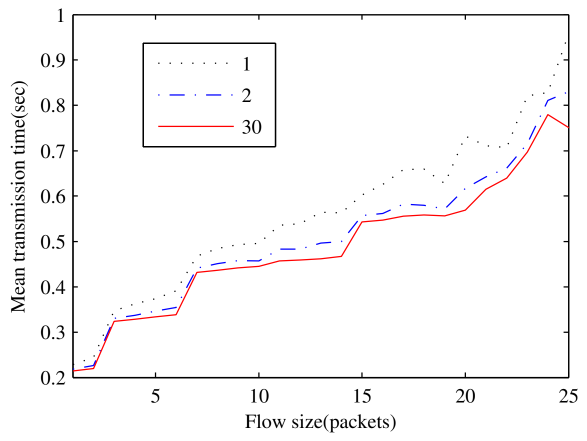

Fig. 9. The impact of quantum ratio on short flows

performance metrics include 1) *the mean transmission time* of short and all flows (transmission time of a flow is computed as lifetime of the flow) , 2) the number of *dropped packets* of all flows, and 3) *throughput* of long flows.

Fig. 5 shows the mean transmission time of short flows. Flow size is computed as the received good packets, excluding the good duplicated packets. It is discovered the mean transmission time of flows with flow size less than the threshold under DRR–SFF is almost the same as that under LAS . But LAS shows better performance for short flows larger than the threshold. From Fig. 5, we observe both DRR–SFF and LAS significantly reduce the mean transmission time of short flows compared with FIFO. As LAS is strict SFF, the mean transmission time of short flows is minimum under LAS. The result shown in Fig. 5 matches with the ideas behind SFF and DRR–SFF.

Fig. 6 depicts the mean transmission time of all flows. We find that the mean transmission time of the very long flows under DRR–SFF is less than that under LAS. The delay of a long flow with 5617 packets under LAS is 207.86 seconds, while that under DRR–SFF is 153.76 seconds. DRR–SFF improves about 25%.

Next, we show the number of dropped packets for all flows in Fig. 7. It is clear that long flows experience less drop rate under DRR–SFF than under LAS.

To show the throughput of long flows, we add a long–lived FTP flow. Fig. 8 shows the FTP flow's throughput by the number of received packets per seconds (counting every 10 seconds). We start counting the received packets after the TCP enters *congestion avoidance* phase. From Fig. 8, we can see the throughput in LAS is almost zero for a period. This is because the long flows are penalized and starved. However, the throughput under DRR–SFF is closed to that under FIFO and the performance of long flows is maintained.

From the dropped packets and throughput of long flows, we can conclude DRR–SFF treats long flows more fairly than LAS does. Moreover, the performance of short flows under DRR–SFF is very close to that under LAS.

Finally, we study the impact of *quantum ratio*. Fig. 9 shows the mean transmission time of short flows using different quantum ratio values. As shown in Fig. 9, large *quantum*

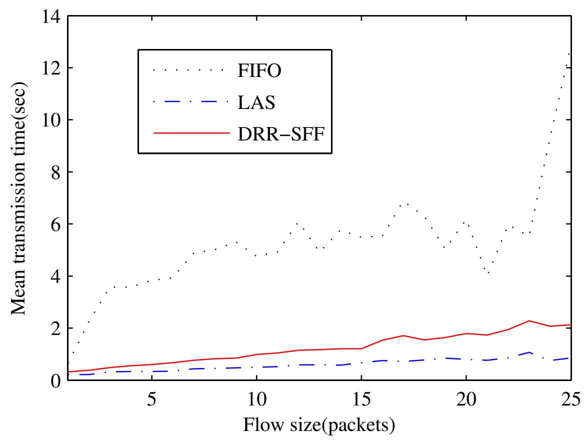

Fig. 10. The mean transmission time of short flows in special case

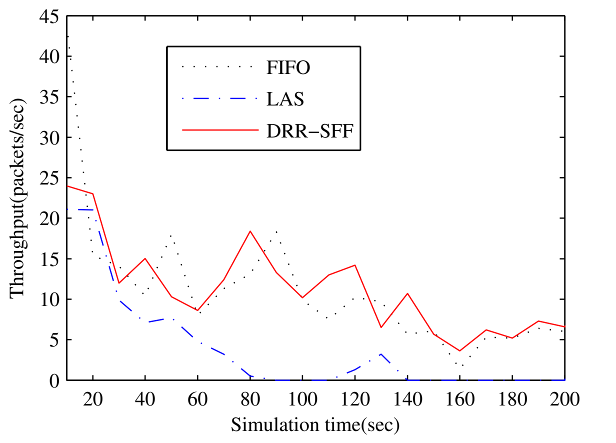

Fig. 11. Throughput of a long flow in special case

*ratio* value (30) is not needed to improve the performance of short flows as the performance of short flows using *quantum ratio* 30 is close to that using *quantum ratio* 2. More complex experiments using different flow size distributions show the same results. Then we can use small quantum ratio value (around 2) to improve performance of short flows and also to guarantee the fairness of long flows.

### *C. Special Case*

If short flows contribute more bytes than normal condition, strict SFF would starve long flows too much. To simulate this special case, we add more short flows to the throughput experiment, and make short flows contribute nearly 40% of total network load. Fig. 10 shows the mean transmission time of short flows, which is slightly increased under DRR–SFF compared with that under LAS. Besides, the throughput of the long FTP flow is shown in Fig. 11. The long flow is starved almost during the whole simulation time under LAS, while its performance under DRR–SFF is close to, some times better than, that under FIFO.

## V. Conclusion

We propose a new short flow first algorithm, DRR–SFF, with *O(1)* computation complexity in order to deploy it in the edge/access routers and improve performance of short flows without penalizing long flows much. Different from previous strict SFF approaches, short and long flows in DRR–SFF are scheduled using Weighted Deficit Round Robin.

The advantage of DRR–SFF over SFF approaches lies in 1) Previous SFF approaches are hard to be deployed especially by hardware; 2) Strict SFF approaches are unfair to long flows and may starve them.

Trace–driven simulation shows that the mean transmission time of short flows under DRR–SFF is almost the same as that under LAS(SFF). On the other hand, the performance of very long flows under DRR–SFF is also held close to that of long flows under FIFO. Besides, the performance of very long flows is improved about 25% compared with LAS(SFF).

## Acknowledgment

We thank Dr. Rai for providing the ns script of LAS, Dr. Ayesta and Dr. Avrachenkov for discussing the ns evaluation of SFF, Yachao Zhou for valuable suggestions, and the anony– mous reviewers for their comments.

## References

[1] L. Guo and I. Matta, "The war between mice and elephants," in *Proc. IEEE ICNP*, 2001.
[2] X. Chen and J. Heidemann, "Preferential treatment for short flows to reduce web latency," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 41, no. 6, pp. 779–794, 2003.
[3] K. Avrachenkov, U. Ayesta, P. Brown, and E. Nyberg, "Differentiation between short and long TCP flows: predictability of the response time," in *Proc. IEEE INFOCOM*, March 2004.
[4] I. A. Rai, E. W. Biersack, and G. Urvoy–Keller, "Size–based scheduling to improve the performance of short TCP flows," *IEEE Network*, January/February 2005.
[5] M. E. Crovella and A. Bestavros, "Self–similarity in World Wide Web traffic: evidence and possible causes," *IEEE/ACM Trans. Networking*, vol. 5, no. 6, pp. 835–846, December 1997.
[6] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot, "Packet–level traffic measurements from the Sprint IP backbone," *IEEE Network*, vol. 17, no. 6, pp. 6–16, 2003.
[7] F. D. Smith, F. Hernndez–Campos, K. Jeffay, and D. Ott, "What TCP/IP protocol headers can tell us about the web," in *Proc. ACM SIGMETRICS*, June 2001, pp. 245–256.
[8] A. Broido, Y. Hyun, R. Gao, and k. claffy, "Their share: diversity and disparity in IP traffic," in *Proc. PAM*, 2004.
[9] "Microsoft security bulletin (ms99–046)." [Online]. Available: http://www.microsoft.com/technet/security/bulletin/ms99–046.mspx
[10] "SSAC advisory SAC008 DNS Distributed Denial of Service(DDoS) attacks," 2006. [Online]. Available: http://www.icann.org/committees/security/dns–ddos–advisory–31mar06.pdf
[11] M. Shreedhar and G. Varghese, "Efficient fair queuing using Deficit Round Robin," *IEEE/ACM Trans. Networking*, vol. 4, no. 3, pp. 375–385, June 1996.
[12] "Cisco: MDRR overview." [Online]. Available: http://www.cisco.com/warp/public/63/mdrr_wred_overview.html
[13] P. E. McKenney, "Stochastic fairness queueing," in *Proc. IEEE INFOCOM*, 1990.
[14] K. C. Claffy, H.–W. Braun, and G. C. Polyzos, "A parameterizable methodology for Internet traffic flow profiling," *IEEE J. Select. Areas Commun.*, vol. 13, no. 8, pp. 1481–1494, October 1995.
[15] "Amcc." [Online]. Available: www.amcc.com
[16] "ns–2." [Online]. Available: http://www.isi.edu/nsnam/ns/
[17] "Tsinghua dragonlab." [Online]. Available: http://dragonlab.org/traffic/