Controlling Bursts in Best-effort Routers for Flow Isolation

Miguel A. Ruiz-Sánchez
INRIA Sophia Antipolis France, UAM Iztapalapa Mexico City
mruiz@sophia.inria.fr

Walid Dabbous
INRIA Sophia Antipolis, France
dabbous@sophia.inria.fr

Abstract

In today's Internet a user can be adversely affected by other users that overload the router. To address this problem, routers need to provide flow isolation. In this paper, we present MuxQ, a new queue management mechanism that provides a high degree of isolation without using per-flow queuing. MuxQ protects the multiplexing function of the router buffer by progressively controlling the allocation of buffer space in a FIFO queue. The allocation decision is based on state information of only a limited number of flows: the flows that do currently have packets in the queue. We evaluate MuxQ by simulation and show that it performs vastly better than the classical Drop-Tail. By using a very simple algorithm MuxQ provides reasonable flow isolation.

1. Introduction

In the Internet, resources are shared by using statistical multiplexing, which results in a network service known as the best-effort service. In case of overload, the statistical multiplexing technique degrades the network service in the form of packet delay or packet loss. While service degradation is inherent in statistical multiplexing based systems, in the Internet, users can suffer high service degradation in case of sustained network overload, a state known as congestion. Traditionally, to control congestion in the Internet, sources use the TCP algorithms [7] to discover the available resources and adapt their traffic pattern dynamically. Unfortunately, in today's Internet, responding to congestion is rather a user's choice and in general there are responsive as well as unresponsive users. In fact, responsive applications are penalized because unresponsive flows, intentionally or unintentionally, abuse the cooperative nature of responsive traffic. Other penalized flows are the short-lived flows that have only a small amount of data to transfer. In summary, today's Internet suffers from the lack of isolation among flows. Providing flow isolation is important because it allows that performance perceived by users does not depend on the good behavior of other users. Routers can provide flow isolation by allocating its resources, specially in times of overload. In this paper, we describe MuxQ (Multiplexing Queuing), a novel queue management mechanism to provide flow isolation. We first study the buffering functionality of IP routers. We find the desired properties of a router buffer system then we design a mechanism based on these characteristics. We emphasize that buffers in routers have two functions: A multiplexing function and a burst absorbing function. MuxQ is based on the idea of protecting the multiplexing function from the burst absorbing function by progressively and dynamically controlling the allocation of buffer space in a FIFO queue.

The rest of the paper is organized as follows. Section 2 focuses on the buffering functions of routers. This will allows us to analyze and compare, in section 3, existing schemes to allocate router resources. Section 4 describes MuxQ, our queue management mechanism. Section 5 shows performance measurements. Finally, we present in section 6 conclusions and future work.

2. The functions of router buffers

A main observation is that a buffer in a router is used for two purposes. First, it allows to multiplex simultaneous flows into the output link. Second, it allows to absorb bursts from individual flows. Both functions are affected by sustained overload. While we are particularly interested in protecting the multiplexing function, both functions are important for a router to offer a good service. For routers the only way to control a sustained overload is by dropping packets. Hence, two important decisions for a router are when to drop packets and which packets to drop. By controlling when to drop packets, a router can improve the link utilization and its capacity to absorb transient overloads. By selecting packets to drop, the router can control the buffer occupancy distribution among flows and thus indirectly control the bandwidth allocation. Hence, we can state the next two desired properties for an ideal buffer system:

1. An ideal buffer system must always have free buffer space to absorb transient overloads, in the form of new

Proceedings of the Eighth IEEE International Symposium on Computers and Communication (ISCC'03)
1530-1346/03 $17.00 © 2003 IEEE

flows or in the form of transient bursts from individual flows.

2. An ideal buffer should select packets to drop in such a way that loss distribution follows the buffer occupancy among flows.

3. Discussion of existing resource allocation schemes in routers

Currently, the Internet's best-effort service is provided by FIFO scheduling and Drop-Tail queue management in routers. Routers transmit packets in the order they arrive (FIFO) to the output port queue and when the buffer space is exhausted packets are simply dropped (Drop-Tail). In other words, routers cannot control when to drop packets nor which packets to drop; instead, both decisions are controlled by the users' behavior. As a result, with Drop-Tail, the multiplexing function cannot be protected in the presence of unresponsive flows because these flows can occupy all the buffer space and starve responsive flows by simply sending fast enough. With Drop-Tail, the only way to protect the multiplexing function of buffers is by having all end-systems to control their traffic. In particular, by using the TCP protocol.

RED [4] was proposed to help to avoid congestion. The major improvement of RED, with respect to Drop-Tail, is that space to absorb transient overload is made available (first property of an ideal buffer system). RED achieves this by dropping packets in case of incipient congestion. Another improvement of RED is that it uses randomization to select packets to drop. While RED randomization allows routers to better distribute losses among competing flows, RED randomization cannot avoid that flows using less than their fair share lose packets. Hence, responsive flows not using their fair share are prevented from reaching it and unresponsive users can still starve responsive flows with RED [5]. As a result, RED allows the multiplexing function of buffer to be protected only if all users are responsive. A modified version of RED called FRED was proposed in [5]. By using per-active-flow accounting FRED provides better isolation from aggressive flows than RED.

A different way to allow routers to select packets to drop is by including flow rate information in the packets. With this information, routers can decide if a packet is accepted or dropped depending on the level of overload. An example of this approach is CSFQ[10]. In CSFQ, dropping is function of the flow rate information of the packet and of an estimated fair share rate. Other examples of similar schemes are [1, 3]. The main improvement of this kind of approaches is that the multiplexing function of buffers is protected by avoiding that flows using less than their fair share suffer packet losses. Nevertheless, this kind of approaches requires to insert additional information in packets and more importantly it requires that all edge routers (or end-hosts) in the system agree on a single scheme to consistently label packets.

A completely different way to protect the multiplexing function of router buffer is by maintaining a separate FIFO queue for each flow [6, 8]. Queues are serviced in a Round Robin order. In this case the multiplexing function is decoupled from the burst absorbing function. Since each flow has a different queue, the bursts or traffic pattern of each flow will not disturb the others flows. The problem with this approach is the difficulty of determining the number of active flows at a given moment. Additionally, state per flow must be maintained and complex scheduling is needed. Also, no FIFO scheduling introduces unnecessary delay for low-bandwidth flows with short bursts arrival [2].

4. Our scheme

4.1. Overview

We propose a new scheme that provides isolation of flows for best-effort traffic, without requiring per-flow queuing. Our scheme is based on the main idea that the multiplexing function of a router buffer must be protected from its burst absorbing function. We call our scheme MuxQ for Multiplexing Queuing. As was stated in section 2, to protect the multiplexing function, routers must provide buffer space to absorb transient overloads. It also must drop or accept packets according to the buffer occupancy distribution of flows. To provide buffer space to absorb transient overloads, MuxQ controls the queue length while allowing high throughput and high link utilization. Also, MuxQ progressively controls the allocation of buffer space in a FIFO queue. The allocation decision is based only on state information of a limited number of flows: the flows that do currently have packets in the queue.

MuxQ has a FIFO queue called MUXqueue, see Figure 1. Although this queue works essentially in the same way that the FIFO queue in traditional IP routers, we control the buffer space of the MUXqueue in such a way that in case of overload, only a limited number of packets from each active flow is accepted. For each flow the maximum number of packets that the router accepts is function of the number of active flows and the flow's buffer occupancy.

4.2. Detailed operation

We explain our mechanism by starting from the traditional FIFO Drop-Tail router then we will refine the scheme to achieve our goals.

In traditional IP routers with drop tail, the buffer occupancy distribution among flows is not controlled by the

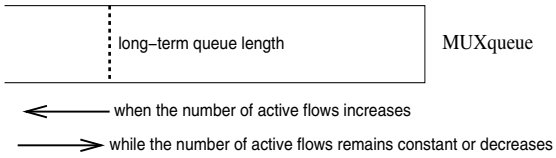

Figure 1. The MuxQ queue

router but by the users' behavior. Among other problems this can keep the buffer full, because packets are dropped only when the buffer space gets exhausted.

As a result, a packet from a new flow may be refused while other flows have a rather large number of packets accepted. In other words, the multiplexing capacity is completely reduced because some flows have already used buffer space to absorb their bursts. To address this problem MuxQ controls the allocation of buffer space in such a way that: First, buffer space is always made available to accommodate packets from new flows. The idea is to control the queue length in such a way that in the long term the queue has a smaller length than the buffer size. We will refer to this length as *ltqlen* (long-term queue length), see Figure 1. Second, buffer space is progressively shared among the active flows. We explain this with an example.

Suppose the MUXqueue is empty and a packet arrives. At this moment there is only one active flow and this flow is restricted to have a maximum number of packets (denoted *maxpkts*) equal to *ltqlen*, in the MUXqueue at the same time. Since *ltqlen* is set to a value smaller than the buffer size, a packet from a second flow will always be accepted. Suppose that by the time when the packet from the second flow arrives the first flow has already *ltqlen* packets enqueued. At this moment the maximum number of packets allowed per flow will be reduced to *maxpkts*=*ltqlen*/2 because the number of active flows has changed to 2. Note that actually the first flow has more packets enqueued than the maximum allowed. Nevertheless, this is a transient overload because the first flow will not be allowed to enqueue further packets until its packet population gets smaller than the new *maxpkts* value. On the other hand, the second flow can continue to enqueue packets. Clearly if no other flow appears the queue length will tend to the *ltqlen* value, with half of the space for each flow (assuming flows continue to send packets). But suppose that another packet from a third flow appears. In this case the number of active flows will be 3 and the maximum number of packets allowed per flow will decrease to *maxpkts*=*ltqlen*/3. Again flows with a packet population exceeding the maximum allowed will not have packets accepted for some time while flows with a number of packets below this maximum will continue to enqueue packets. Clearly, more the number of new flows more the greedy flows will be penalized.

In summary, in the long term, the queue length is maintained shorter than the buffer size and tends to the value *ltqlen*. As we can see, a dynamic reserved part of the buffer is always available to accommodate new flows. This part of the buffer is dynamically delimited by the *ltqlen* value and the maximum buffer size. We protect in this way the multiplexing function of the router buffer. Controlling the long-term queue length has the additional benefit of controlling the long-term delay of packets. The MuxQ algorithm is shown in Figure 2.

```
procedure Enqueue {
  check number of packets (npkts) from this flow already
  in the MUXqueue;  /* by hashing */
  if(packet is from a new active flow, i.e. npkts==0) then {
    increment number of active flows;
    set npkts=1 in the hash table entry for this flow;
    insert packet in the MUXqueue;
    update maxpkts=ltqlen/number of active flows;
  } elseif (npkts < maxpkts) then {
    increment npkts in the hash table entry for this flow;
    insert packet in the MUXqueue;
  } else
    drop packet;
}

procedure Dequeue {
  serve packet in the front of the MUXqueue;
  decrement npkts in the hash table entry for the
  flow the packet belongs to;
  check number of packets (npkts) from this
  flow still in the MUXqueue;  /* by hashing */
  if (npkts== 0 ) then {
    decrement number of active flows;
    update maxpkts=ltqlen/number of active flows;
  }
}
```

Figure 2. The MuxQ algorithm

As we have seen, *maxpkts* the maximum number of packets in the buffer from the same flow, will be reduced according to the number of active flows. This allows to share the buffer space among the competing flows. More the number of competing flows less the buffer space allowed per flow.

With MuxQ the bursts are naturally accepted as long as their size do not trouble the multiplexing capacity (*maxpkts*). In case the aggregated bursts size (number of packets in the MUXqueue) for an active flow is larger than *maxpkts,* its new incoming packet is dropped. Note that packets are dropped also if a very large number of new flows arrives simultaneously at the MUXqueue.

4.3. Active flows

Since routers treat packets independently, the concept of flow is not natural at the router level. Nevertheless, some notion of flow is necessary if we want to protect the multiplexing function. This notion must be more related to a dynamic soft-state than to the real end-to-end flow notion. In fact, in our scheme the notion of flow is associated to the group of packets with some defined specific subset of identifiers and which are in the MUXqueue at a given moment. In other words, a flow in our scheme lasts as long as the flow has packets in the MUXqueue.

When a packet arrives at the router, an address lookup operation is performed and then the packet is directed to the appropriate output link. Then it is checked if packets from the same flow are already in the MUXqueue. To make this operation efficient we use a hashing table containing the number of packets of each flow in the MUXqueue. It is worth to note that state information is maintained only for the flows that do have packets in the main buffer at any moment. The table is updated when a packet arrives and when a packet leaves the router using a hashing operation, see Fig.2.

5. Performance

In this section, we evaluate our queue management mechanism by simulation. More specifically, we study the ability of MuxQ to isolate different kind of flows. The performance of MuxQ is compared with those of Drop-Tail, CSFQ, FRED, and DRR by using the ns-2 simulator, which we extended with a MuxQ module. CSFQ and FRED simulation code was obtained from [9]. DRR and Drop-Tail are included in the standard ns-2 package. Since DRR uses per flow queuing, nearly perfect isolation can be obtained with this mechanism; it is used as a reference in our simulations. We present the results of several simulation experiments, each of which focuses on a particular traffic condition. Both TCP sources and constant bit rate (CBR) UDP sources are used. Our simulations use the topology shown on figure 3. The bottleneck link bandwidth is 10 Mbps and the link's propagation delay is 5 ms. The bottleneck router has a maximum buffer size of 100 KBytes and all packets are 1000 bytes long. For MuxQ the ltqlen value was fixed at 0.75 of the buffer size.

5.1. Protecting long-lived responsive flows (TCP) from each other.

In this first experiment we seek to show that our mechanism does not interfere with the end-to-end TCP congestion algorithm. For network traffic, we use FTP transfers over TCP. Each source initiates an FTP transfer at the beginning

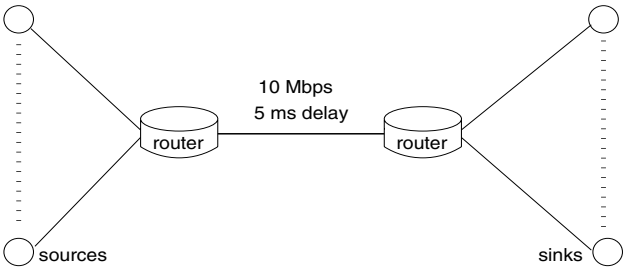

Figure 3. The simulation topology

of the simulation and continues until the end of the simulation. Figure 4 shows the average throughput achieved by each flow over a 50 sec interval. As we can see, MuxQ provides almost perfect isolation. DRR provides also almost perfect isolation, but DRR uses per flow queuing which is not the case for MuxQ. For this traffic, Drop-Tail provides reasonable isolation. This is because all sources are responsive and all have the same round-trip time. For CSFQ and Fred the throughput of flows shows more variability.

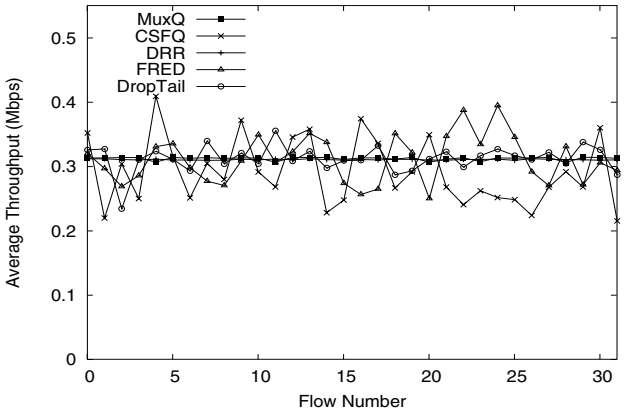

Figure 4. Average throughput of 32 TCP flows sharing a 10 Mbps link.

5.2. Protecting long-lived responsive flows (TCP) from non responsive flows

In this experiment, we examine MuxQ's ability to isolate responsive flows from non responsive flows bad effects. Five of the TCP flows in the last experiment are substituted with 5 aggressive CBR flows (flows 27 to 31) sending at 10 Mbps each one. Figure 5 shows the average throughput of each flow over a 50 sec interval. Again DRR performance is almost perfect while Drop-Tail does not protect at all the

TCP flows from the aggressive CBR flows. Performance of MuxQ is slightly better than that of CSFQ.

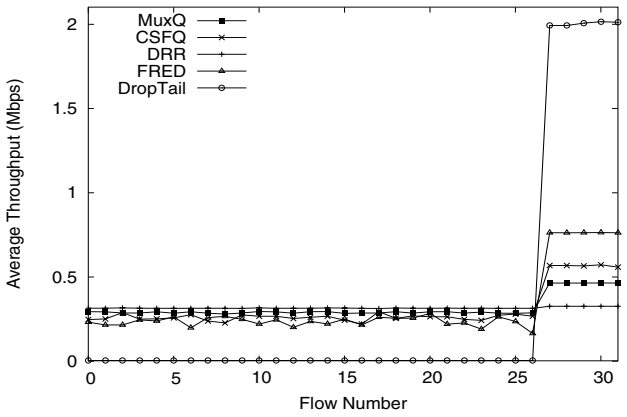

Figure 5. Performance degradation of several TCP flows (flows 0 to 26) when competing with 5 aggressive CBR flows (flows 27 to 31).

In another experiment, a TCP flow competes with an increasing number of CBR flows. The TCP flow is generated by a source which always has data to send, and the CBR flows are generated by unresponsive sources which transmit packets at a constant rate of twice its fair share. In Figure 6 we show the normalized average throughput achieved by the TCP flow. The results show again that with Drop-Tail the performance of the TCP flow is severely degraded even with only one CBR flow. DRR provides almost perfect isolation. With MuxQ and CSFQ the performance of the TCP flow is maintained at a reasonable level.

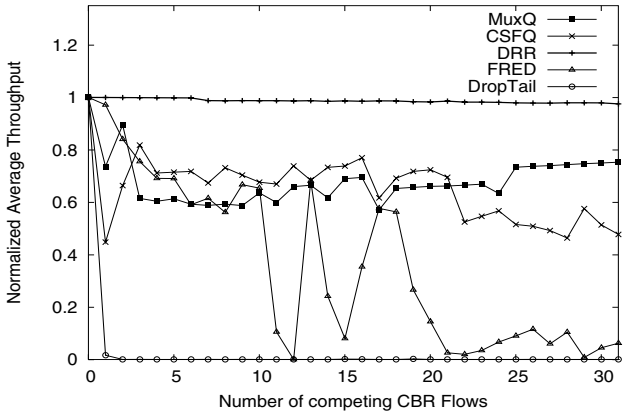

Figure 6. Throughput of 1 TCP flow competing with an increasing number of CBR flows

5.3. Protecting short-lived flows

In this experiment, short-lived web-like TCP flows compete with long-lived flows: 5 long-lived TCP flows and 1 aggressive CBR flow sending at 10 Mbps. For the long-lived flows, each source initiates its transfer at the beginning of the simulation and continues until the end of the simulation, i.e 50 sec. 20 web clients make random requests to a web server. The server responds to each request by sending a web page with one object of 12 Kilobytes. In other words, all web-like flows are of the same size, i.e. 12 Kilobytes. The time between retrieval of two successive pages follows an exponential distribution with mean equals to 3 sec. Figure 6 shows the cumulative distribution of the web-like flows. Horizontal axis is the time to complete the flow transmission. The figure shows the number of finished web-like flows after 50 sec of simulation. We can observe that with MuxQ most of the web-like flows have smaller response times than with the other schemes. Note also that for the same simulation time, the number of successful finished flows is different for each scheme. MuxQ outperforms in this aspect to DRR and Drop-Tail. Unfortunately, we were not able to run this simulation nor with CSFQ nor with Fred with the source code from [9].

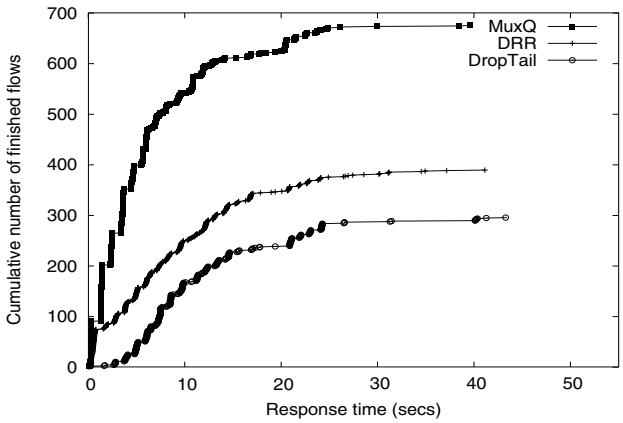

Figure 7. Web-like traffic competing with 5 long-lived TCP flows and 1 aggressive CBR flow sending at 10 Mbps.

5.4. How non-responsive flows are affected each other

In this experiment, we seek to show how unresponsive flows with different rates are affected each other. CBR flow number i sends packets at a rate (i+1) times its fair share. In other words, flows transmit at different rates which go

from the fair share rate to the maximum link rate. Note that all flows but the first one send packets at a rate greater than its fair share rate. Figure 8 shows the average performance of the different flows. As we can see, Drop Tail does not isolate flows. Flows which transmit at a larger rate get more bandwidth than lower rate flows. MuxQ, DRR, and CSFQ provides almost perfect isolation. Fred performance is only slightly better than that of Drop-Tail.

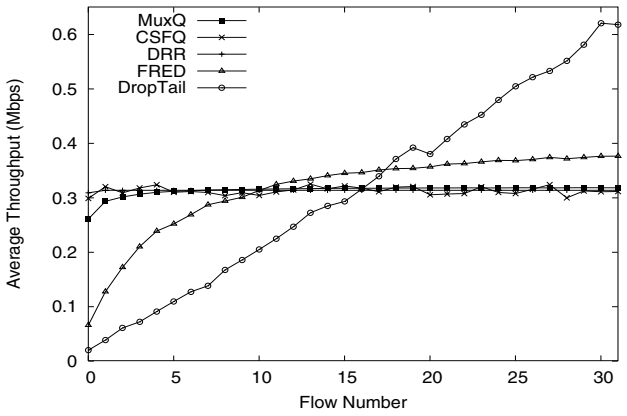

Figure 8. Average throughput of 32 CBR flows sending at different rates. Flow i sends packets at a rate (i+1) times its fair share.

5.5. MuxQ deployability

Even though users can choose to not use TCP as their transport protocol, it is a fact that TCP remains highly utilized. We have shown that MuxQ behaves very well when TCP flows cross a MuxQ router. Also, MuxQ does not need modifications to the IP header. More importantly, MuxQ does not require that hosts or end-routers agree on a single scheme to consistently include flow information. In other words, MuxQ does not expect a special behavior from routers nor from hosts. Thus, MuxQ routers can be deployed incrementally in the Internet.

6. Conclusion

We have designed a new queue management mechanism for flow isolation. Our mechanism is based on the idea to protect the multiplexing function from the burst absorbing function of router buffers. MuxQ which is based on a FIFO queue, uses a very simple algorithm to allocate buffer space and control the queue length.

We compared the performance of the proposed scheme to that of classical Drop-Tail and to that of other proposed schemes, including CSFQ and DRR which provides nearly perfect isolation by using per-flow queuing. By keeping only limited flow-state our mechanism performs very much better than Drop-Tail. MuxQ achieves performance similar to that of CSFQ but MuxQ does not need modifications to the IP packet header as it is the case for CSFQ.

One of the important characteristic of a new router mechanism is its incremental deployability. MuxQ does not need modifications of the IP packet header. Moreover, since MuxQ does not expect a special behavior from other routers, MuxQ routers can interact without problem with classical Drop-Tail routers and thus MuxQ can be deployed incrementally. We believe that MuxQ is an interesting approach to achieve a high degree of flow isolation with respect to Drop-Tail by using a very simple algorithm.

We intend to study in future work how buffer size affects the performance of MuxQ. Another aspect that deserves further investigation is how to preserve isolation on multiple congested links.

References

[1] Z. Cao, Z. Wang, and E. W. Zegura. Rainbow fair queueing: Fair bandwidth sharing without per-flow state. In *Proceedings of INFOCOM 2000*, pages 922–931, 2000.

[2] D. D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet network: architecture and mechanism. In *Proceedings of ACM SIGCOMM'92*, pages 14–26, Aug. 1992.

[3] A. Clerget and W. Dabbous. Tuf : Tag-based unified fairness. In *Proceedings of INFOCOM 2001*, pages 498–507, Apr. 2001.

[4] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[5] D. Lin and R. Morris. Dynamics of random early detection. In *Proceedings of ACM SIGCOMM'97*, pages 127–137, 1997.

[6] J. Nagle. On packet switches with infinite storage. *IEEE Trans. Communications*, 35(4):435–438, Apr. 1987.

[7] J. Postel. Transmission control protocol. *RFC 793*, Sept. 1981.

[8] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *Proceedings of SIGCOMM'95*, pages 231–242, 1995.

[9] I. Stoica. Csfq and fred ns-2 simulation source code. In *at: http://www.cs.berkeley.edu/ istoica/csfq/*.

[10] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks. In *Proceedings of SIGCOMM'98*, pages 118–130, 1998.