Rainbow Fair Queueing: Fair Bandwidth Sharing Without Per-Flow State

Zhiruo Cao † *Zheng Wang* ‡ *Ellen Zegura* †

† College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
{zhiruo,ewz}@cc.gatech.edu

‡ Bell Labs
Lucent Technologies
Holmdel, NJ 07733
zhwang@dnrc.bell-labs.com

Abstract—**Fair bandwidth sharing at routers has several advantages, including protection of well-behaved flows and possible simplification of end-to-end congestion control mechanisms. Traditional mechanisms to achieve fair sharing (e.g., Weighted Fair Queueing, Flow Random Early Discard) require per-flow state to determine which packets to drop under congestion, and therefore are complex to implement at the interior of a high-speed network. In recent work, Stoica et al. have proposed Core-Stateless Fair Queueing (CSFQ), a scheme to approximate fair bandwidth sharing without per-flow state in the interior routers. In this paper, we also achieve approximate fair sharing without per-flow state, however our mechanism differs from CSFQ. Specifically, we divide each flow into a set of layers, based on rate. The packets in a flow are marked at an edge router with a layer label (or "color"). A core router maintains a color threshold and drops layers whose color exceeds the threshold. Using simulations, we show that the performance of our Rainbow Fair Queueing (RFQ) scheme is comparable to CSFQ when the application data does not contain any preferential structure. RFQ outperforms CSFQ in goodput when the application takes advantage of the coloring to encode preferences.**

Keywords—**Fair queueing, differentiated services, layering.**

I. Introduction

FAIR bandwidth sharing at routers protects well-behaved flows from mis-behaving flows and may simplify or enhance end-to-end congestion control mechanisms. Three types of mechanisms can be used to achieve fair bandwidth sharing. *Per-flow queueing mechanisms* (e.g., [1], [2]) operate by maintaining a separate queue for each flow and performing FIFO ordering with tail-drop on each queue. *Per-flow dropping mechanisms* (e.g., Flow Random Early Drop (FRED) [3]) employ a single FIFO queue, but maintain per-flow state to determine which packets to drop under congestion. Because both of these mechanisms require per-flow state, their suitability for use over very high speed backbone trunks with a large number of flows has been questioned.

Recently, Stoica et al. have proposed a scheme that approximates fair bandwidth sharing without requiring per-flow state in the core routers [4]. The main idea behind their Core-Stateless Fair Queueing (CSFQ) architecture is to keep per-flow state at slower edge routers and carry that information in packets to the core. Specifically, packets are labeled with flow arrival rate; core routers estimate the fair share and probabilistically drop packets whose arrival rate (as marked) exceeds the fair share. Through extensive simulations, CSFQ is shown to achieve a reasonable degree of fairness: CSFQ tends to approach the fairness of Deficit Round Robin (DRR) [5] and to offer considerable improvement over FIFO or Random Early Detection (RED) [6]. CSFQ offers great promise for making fair bandwidth sharing feasible in high-speed networks.

In this paper, we also aim to achieve approximate fair sharing without per-flow state. Our approach is a combination of a color labeling scheme and a buffer management mechanism. We divide each flow into a set of layers, with a globally consistent rate per layer. The packets in a flow are each marked at the edge router with a label that reflects the "color" of the layer. A core router maintains a color threshold C; packets with a color label larger than C are dropped. During congestion, the color threshold is decreased; when congestion clears, the color threshold is increased. Because the coloring is based on rate, the discarding of packets is approximately fair. We call our scheme Rainbow Fair Queueing (RFQ), due to the use of "colors" to convey rate information.

The main differences between RFQ and CSFQ are:

- The state information carried by the packets are the color layers they belong to, rather than the explicit rate of their flows.
- The operation of the core routers are further simplified. Note that the fair share calculation in CSFQ requires exponential averaging to estimate the input rate. In RFQ, the core routers only need to perform threshold-based dropping. The scheme is quite simple and amenable to hardware implementation.
- An application can express a preference for certain packets to be preserved under congestion by marking them with a lower color value, subject to constraints on the rate for each color.

The simulation results demonstrate that the performance of RFQ is comparable to CSFQ when the application data does not contain any preferential structure. RFQ outperforms CSFQ and DRR in goodput[1] when the application takes advantage of the coloring to encode preferences.

The remainder of the paper is structured as follows. In the next section, we discuss related work in queue management and layering. In Section III we describe the basic operation and details of RFQ. In Section IV we evaluate the performance of RFQ in comparison to other schemes, including DRR, RED and CSFQ. In Section V, we discuss how layered applications encode preferences in colors and study the performance improvement. Finally, we conclude in Section VI.

[1]Goodput is the effective throughput, as determined by the data successfully received and decoded at the receiver. We will discuss the goodput of video flows in greater detail in Section V.

II. Related Work

In the past ten years, Weighted Fair Queueing (WFQ) has attracted considerable attention as a mechanism to achieve fair bandwidth sharing and delay bounds [1], [2]. Strict WFQ is, however, generally considered too complex to implement in practice. Many variants of the WFQ algorithm have been proposed, with different tradeoffs between complexity and accuracy [5], [7], [8]. Nevertheless, the WFQ computation is only part of the task; a per-flow queueing system also requires flow classification and per-flow state maintenance. Every incoming packet has to be classified to its corresponding queue. The per-flow reservation state has to be installed by a setup protocol such as RSVP and retrieved during packet forwarding for scheduling calculations. All of these are a considerable challenge when operating at high speeds with a large number of flows.

Random Early Detection (RED) has been proposed as a replacement to the simple tail drop [6]. RED monitors the average queue size and probabilistically drops packets when the queue exceeds certain thresholds. By dropping packets before the buffer is full, RED provides an early signal to the end systems to back off. However, RED cannot ensure fairness among competing flows. Flow Random Early Discard (FRED) improves the fairness of bandwidth allocation in RED by maintaining state for any backlogged flows [3]. FRED drops packets from flows that have had many packets dropped in the past or flows that have queues larger than the average queue length.

An interesting research question is, therefore, whether one can approximate the operation of WFQ without requiring the use of per-flow state. Although such approximations may not be able to match the fairness and accuracy of WFQ, the objective is to achieve fair bandwidth sharing with a simpler and scalable approach. Core-Stateless Fair Queueing (CSFQ) does exactly this [4]; we have the same goal.

In addition to fair queueing schemes, our work also draws upon the concept of layering. Layering has been used in the past for congestion control of multicast video and audio streams [9], [10], [11]. In these schemes, video or audio traffic streams are usually divided into a small number of layers, typically through sub-band encoding. Receivers can choose the number of layers of traffic to receive based on the bandwidth available to them. Closely related to our work are efforts to divide traffic of various types into more layers. Such an approach has been considered for video [12] and for bulk-data transfer [13], [14].

III. Rainbow Fair Queueing

In this section, we present RFQ, a packet coloring and buffer management scheme that emulates the fair sharing of WFQ but avoids packet classification and per-flow state operations in the core. With RFQ, core routers still perform FIFO scheduling but with more sophisticated buffer management. RFQ's implementation complexity is much lower compared with a per-flow queueing system.

The network model is the same as that used in CSFQ [4] and in Differentiated Services: namely, a network comprised of edge routers and core routers. (See Figure 1.) The edge routers perform packet classification and encode certain state in packet headers, and the core routers use the state for packet discarding.

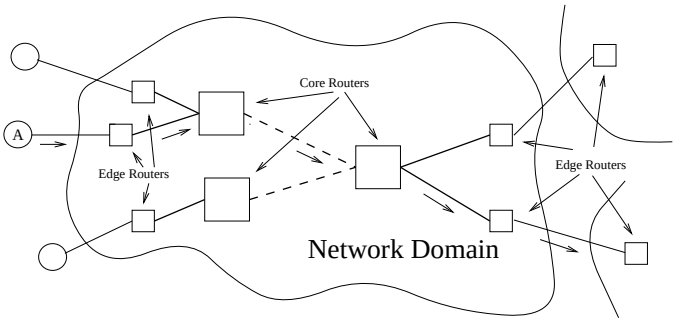


Fig. 1. Network Architecture

In this model, a *flow* is a stream of packets which traverse the same path in a network domain and require the same grade of service at each router in the path. Next, we describe a version of RFQ assuming that all flows have the same weight. A weighted version of RFQ is presented at the end of this section.

A. RFQ Overview

RFQ has two main components: a packet coloring algorithm for edge routers and a buffer management and packet discard algorithm for core routers. When a flow arrives at the edge of the network, its rate is estimated by an exponential average based on the packet inter-arrival time. The edge router then divides the flow into many thin layers. Each layer is assigned a number, which we call a "color". The colored layers have two purposes. First, they reflect the rate of the flow: the larger the number of colored layers is, the higher the rate of the flow is; flows with the same rate have the same number of colored layers. Second, the colored layers provide a structure for controlled discarding in the network when congestion occurs.

Inside the network, different flows are, of course, interleaved. Since the core routers do not perform per-flow operations, they cannot distinguish packets on a per-flow basis. Instead, they operate on a single packet stream with many different colored layers. The core routers still operate in simple FIFO fashion. When the backlogged packets exceed some threshold, the core routers discard packets to reduce the traffic load. The discarding starts with the packets with the highest color value. That is, the routers "cut" layers of color from top down. Because the coloring is done proportionally to the rate, the packet discarding approximates that of a WFQ system.

Let us illustrate the operation of RFQ with a simple example. Figure 2 shows three flows with rates of 10 Kbps, 6 Kbps and 8 Kbps. We assume that each color layer has a rate of 2 Kbps. (We later discuss the issue of a "good" selection of rates for the layers.) When the three flows arrive at the edge of the network, they are divided into colored layers based on their rates. Thus, flows A, B and C have five, three and four layers respectively. Suppose that the three flows converge at a core router. Then we have a packet stream with five layers and the composition is given in Table I.

When congestion is detected, the core routers will discard layer four first. Note that all packets in colored layer four come from flow A. Thus, the rate of flow A is reduced to 8 Kbps, and flows B and C are not affected at all. Suppose that the bottleneck only has a capacity of 18 Kbps. Layers four and three will

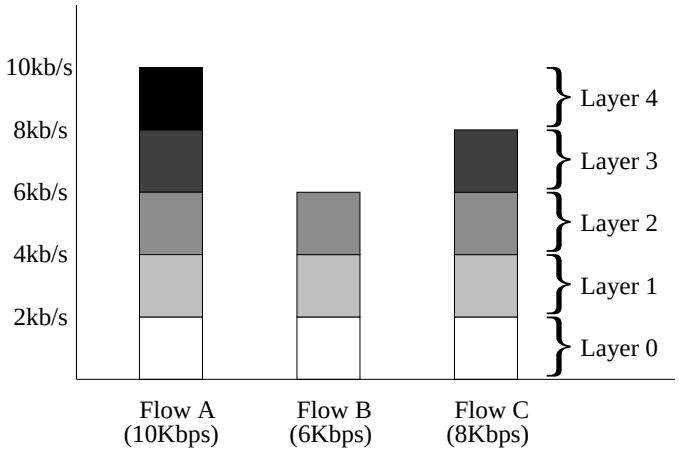


Fig. 2. Example of RFQ Coloring

Color	Rate (Kbps)	Contribution
layer 4	2	flow A
layer 3	4	flow A, B
layer 2	6	flow A, B and C
layer 1	6	flow A, B and C
layer 0	6	flow A, B and C

TABLE I
COMPOSITION OF TRAFFIC IN COLORED LAYERS

then be cut, and all three flows will receive the same amount of bandwidth (6 Kbps). As we can see from this example, the structure provided by the colored layers allows the packet discarding to be done in a way similar to that of WFQ.

B. RFQ Details

This basic mechanism has four important details: (1) the estimation of the flow arrival rate at the edge routers, (2) the selection of the rates for each color, (3) the assignment of colors to packets, and (4) the core router algorithm. We consider each of these details in turn.

B.1 Flow arrival rate estimation

At the edge routers, the flow arrival rate must be estimated, in order to assign a color. To estimate the flow arrival rate, we use the same exponential averaging formula as in CSFQ [4]. This scheme requires an edge router to keep state for each active flow. Specifically, let t_i^k and l_i^k be the arrival time and length of the k^{th} packet of flow i. The estimated rate of flow i, is calculated as:

$$r_i^{new} = (1 - e^{-T_i^k/K})\frac{l_i^k}{T_i^k} + e^{(-T_i^k/K)} r_i^{old}$$

where $T_i^k = t_i^k - t_i^{k-1}$ and K is a constant. Using an exponential weight $e^{-T_i^k/K}$ gives more reliable estimation for bursty traffic, even when the packet inter-arrival time has significant variance.

B.2 Selection of layer rates

After estimating the flow arrival rate, each packet is assigned a color, with the constraint that the average rate of packets with color i is at most c_i. We discuss the issue of assigning colors to packets below; we are concerned here with the choice of the rates c_i. One important fact is that colors with smaller values are assigned before the colors with large values. That is, a flow with arrival rate r will be assigned colors with $0..j$, where j is the smallest value satisfying $\sum_{i=0}^{j} c_i \geq r$.

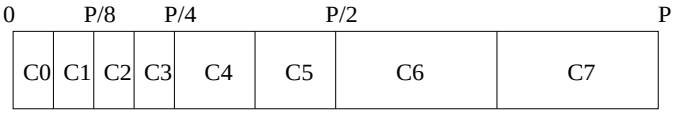


Fig. 3. Flows are partitioned into colored layers according to their rates.

Note that the color label is carried in the packet header, thus the number of values we can use is limited. For IPv4, for example, the color label can be put in the Type of Service field of IP header as it is used in Differentiated Services [15]. If the label is 8-bit long, we have 256 color values.

There are clearly many options for how to select the rates c_i. The simplest approach is to make all color layers have equal rate. However, for low rates, the granularity can be very coarse. For example, assume the rate of a flow's top layer is c_k. When congestion occurs, cutting one layer will reduce the total throughput by c_k/r. When r is small, the layer cut will severely reduce the total throughput of the flow. If there are many such low rate flows sharing the link, a layer cut will cause the link to become severely under-utilized. Therefore, one of the objectives in layer rate selection is to minimize the affect to a flow and link utilization when a layer cut takes place.

Based on simulations and analysis, we use a non-linear encoding in which lower layers are given smaller rates (and thus finer granularity), while higher layers are given larger rates. We divide the rate spectrum into "blocks" and use a combination of equal rates (within a block) and exponentially increasing rates (from one block to the next). We select this particular partition method because it requires very simple computation to determine which layer a packet should belong to. Specifically, layer i has rate c_i:

$$c_i = \begin{cases} \frac{a^{INT(i/b)-N/b}}{b} P & b \leq i \leq N \\ \frac{a^{1-N/b}}{b} P & 0 \leq i < b \end{cases}$$

where N is the total number of colors (layers). Parameters a and b determine the block structure, and N must be a multiple of b. P is the maximum flow rate in the network. For example, when $N = 8$ and $a = b = 2$, the layer rates are illustrated in Figure 3. The rate of the layer is depicted by the width of its rectangle.

B.3 Color assignment

We now turn to the issue of assigning colors to the packets. Recall that the constraint is that the *average rate* of packets with color i is at most c_i. However, if the time scale over which the average is taken is too long, significant unfairness and performance degradation can result. A burst of packets with the same color may cause temporary buffer overflow which can be avoided if the traffic within a color is smoothed. Furthermore, dropping of consecutive packets from a TCP connection will significantly affect the throughput of a TCP flow [16], [6].

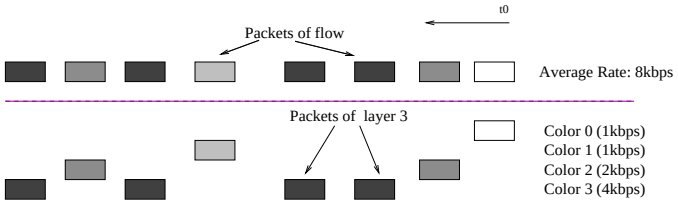


Fig. 4. Color Assignment

There are a number of ways in which colors can be assigned. The simplest is a probabilistic color assignment scheme. Each packet is randomly assigned a color with a probability determined by the layer's rate. Specifically, suppose the current estimate of the flow arrival rate is r, and j is the smallest value satisfying $\sum_{i=0}^{j} c_i \geq r$. Then the current packet is assigned color $0 \leq i \leq j$ with probability $c_i / \sum_{i=0}^{j} c_i$. When all packets have fixed size, it is easy to see that the probabilistic color assignment will cause the rates for each color to approach the required rates c_i. When the packet sizes are variable, it can also be shown that the random assignment will result in average rates for color layers that approximate the designated distribution. An example of color assignment is shown in Figure 4.

Color assignment can also be done with multiple token buckets in a way similar to [17]. Applications can also "pre-color" the packets to reflect preferential treatment. Important packets can be marked with small color values so that they are less likely to be dropped. The packets can keep the color assigned by the applications as long as the total traffic for each layer is within the constraint. We will discuss this aspect in greater detail in Section V.

B.4 Core router algorithm

The core routers monitor buffer occupancy and discard color layers when backlogged packets exceed certain thresholds. The objective is to achieve approximate fair queueing while maintaining high utilization.

The complete core router algorithm is shown in Figure 5. The main task is to update the color threshold C according to the current congestion status. Initially, C is set to the maximum color value.

As shown in the pseudocode, upon each packet arrival, *C_max* is updated to record the largest color value having been seen. When the current queue length *q_len* reaches a threshold *q_threshold*, it is likely that the link is congested and C should be decreased. However, over-reacting may compromise the performance. Our scheme decreases C when the following three conditions are all met: (1) since the last C update, 0.1*q_maxsize* bytes have been accepted into the buffer; (2) the queue length has increased since the last C update; (3) only k consecutive C decreases are allowed, where k is set to be $0.25C$ every time interval *update_int*. Condition 1 prevents very frequent changes to help ensure stability. Condition 2 ensures that the input traffic does exceed the output capacity so that a layer cut is necessary. Condition 3 prevent the system from over reacting on detected congestion. Note that the C decrease frequency is also self-clocked in that when C is set too high, the packet acceptance rate is high, condition 1 is met more quickly, and C is decreased faster.

On packet arrival:
$C\_max = MAX(C\_max, pkt\_color)$;
if (($q\_len > q\_threshold$) and
($b\_rcv \geq q\_maxsize/10$) and
($q\_last < q\_len$) and ($k > 0$)) {
/* Decrease the color threshold */
$C = C - 1$;
$q\_last = q\_len$;
$t\_last = now$;
$b\_rcv = 0$;
$k = k - 1$;
}
else if ($now - t\_last > update\_int$) {
if ($b\_rcv < link\_capacity * update\_int$) {
/* Increase the color threshold */
$C = C + 1$;
$C = MIN(C, C\_max)$;
$q\_last = q\_len$;
$t\_last = now$;
$b\_rcv = 0$;
update($update\_int$);
}
$k = C/4$;
}
/*Count the traffic admitted by the color threshold C*/
if ($C \geq pkt\_color$) $b\_rcv = b\_rcv + pkt\_size$;
/*Determine whether to admit the packet or not */
if (($q\_len == q\_maxsize$) or ($C < pkt\_color$)) {
drop $packet$;
}
enqueue $packet$;

Fig. 5. Core router algorithm

After every time interval *update_int*, if the average input rate *b_rcv/update_int* is less than the link capacity *link_capacity*, the color threshold C is increased. *b_rcv* is the traffic (in bytes) that is allowed into the buffer by the color threshold C. The threshold update interval *update_int* is adjusted every time C is increased so that the closer the service rate is to the link capacity, the longer the *update_int* is and less frequently the color threshold is updated.

C. Weighted RFQ

The RFQ algorithm can support flows with different weights. Specifically, let w_i denote the weight for flow i. Then the packets in this flow are marked such that the average rate for packets labeled with color j is $w_i c_j$. That is, a larger weight allows more packets to be marked with lower color values than a smaller weight. The remainder of the algorithm is essentially the same as the unweighted case.

IV. Simulations

In this section, we present simulation results. In our simulations, we compare the performance of RFQ with the following

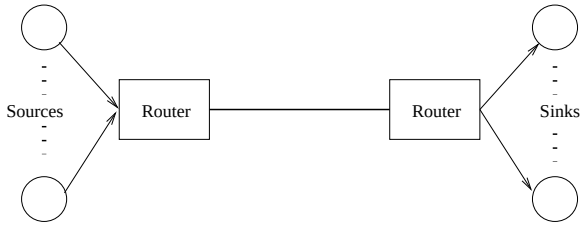


Fig. 6. A single congested link simulation topology. The congested link has capacity of 10 Mbps and 1ms propagation delay.

schemes: Deficit Round Robin (DRR), Random Early Discard (RED) and Core-Stateless Fair Queueing (CSFQ).

DRR is a well-known variant of WFQ that has an implementation complexity of $O(1)$ [5]. In DRR, queues are serviced in a round-robin fashion with a quantum of service assigned to each queue. A packet is dropped from the longest queue when the buffer is full. DRR is the only algorithm that requires per-flow queueing in the four schemes we used for simulation; DRR serves as the benchmark for fair bandwidth sharing.

All simulations were performed using the simulator ns-2 [18]. The simulation code for CSFQ is obtained from [19] and is used unchanged. DRR and RED algorithms as well as TCP and UDP traffic generation algorithms are available with the ns-2 package.

For the sake of better comparison, we used simulation configurations and parameters similar to those in [4]. In all the simulations, unless stated otherwise, the link buffer size is set at 64000 bytes, and the packet size (MTU) is set at 1000 bytes. For video sources, the MTU is set at 500 bytes. The parameters for CSFQ, DRR and RED are set as described in [4]. For RFQ, we set P for all flows at 10Mbps, and the threshold *q_threshold* at 60% of the total buffer size.

A. A Single Congested Link

We first evaluated the performance of RFQ on a simple network configuration as shown in Figure 6. Assume n flows share a single bottleneck link with a capacity of 10 Mbps. We assume that all flows have the same weight. When the link is congested, all backlogged flows should receive the same amount of bandwidth. In the first experiment, each of 32 UDP flows sends at $i \times \frac{10}{32}$ Mbps, where i $(1 \ldots 32)$ is the flow number. During the 10 seconds simulation time, each UDP flow has infinite data to transmit and hence the link is severely congested. Under max-min fairness [1], each flow should achieve an average throughput of 313 Kbps. In Figure 7, we plot the average throughput achieved by each flow when the link is configured using DRR, CSFQ, RED and RFQ. The results show that RED cannot ensure fair bandwidth sharing during congestion while DRR gives almost perfect sharing among contending flows. Both CSFQ and RFQ have similar performance to DRR and perform much better than RED.

Figure 8 shows the comparison of queue dynamics in the all-UDP experiment with RFQ and CSFQ. Since both RFQ and CSFQ use dropping as a way of enforcing bandwidth allocation, it is not surprising that the queue is quite long during the congestion period. From Figure 8, we can see that the queues in RFQ and CSFQ both fluctuate greatly, and the queue length of RFQ is slightly shorter than that of CSFQ.

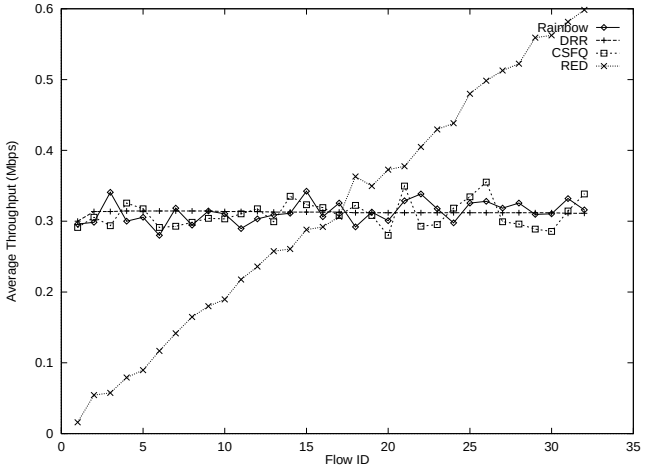


Fig. 7. Average throughput achieved by each of 32 UDP flows sharing a bottleneck link with capacity of 10 Mbps. Each UDP flow i sends at i times its fair share (313 Kbps).

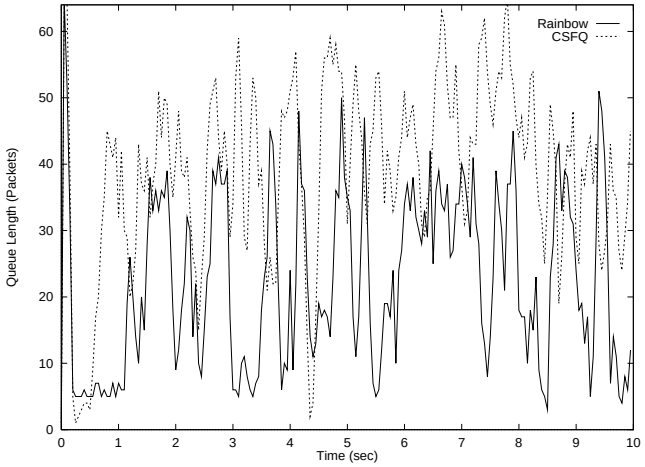


Fig. 8. The queue dynamics in RFQ and CSFQ. In this experiment, 32 UDP flows share a bottleneck link with capacity of 10 Mbps. Each UDP flow i sends at i times its fair share (313 Kbps).

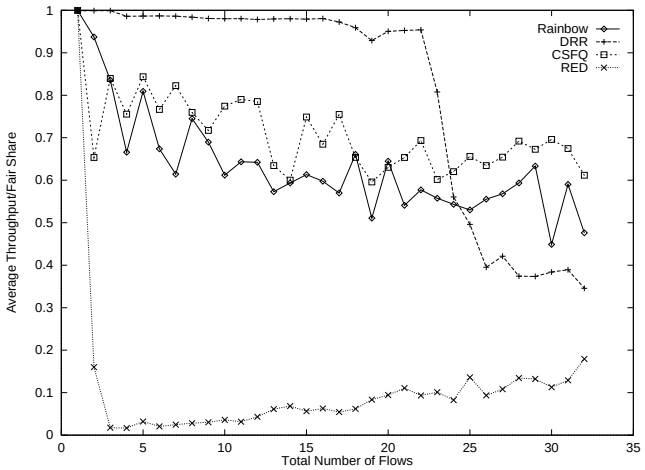


Fig. 9. Average throughput achieved by a TCP flow sharing a bottleneck link of capacity 10 Mbps with $(n-1)$ UDP flows. Each UDP flow sends at twice of its fair share.

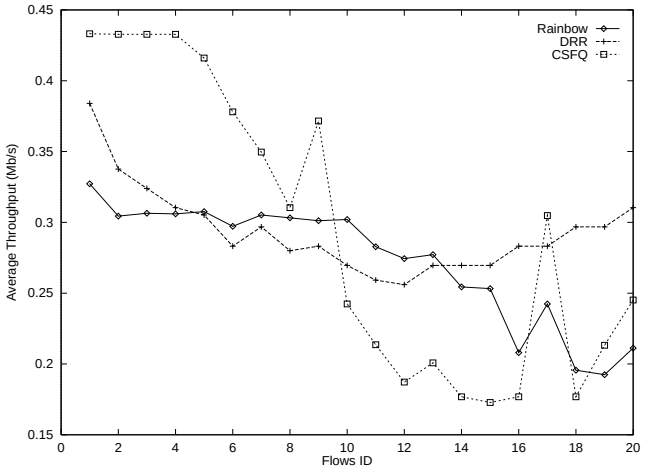


Fig. 10. The average throughput of 20 video sources that share a single bottleneck.

In the UDP experiments, the sources simply send packets at fixed rates thus the results are quite straightforward. Because TCP congestion control will back off during congestion, TCP flows tend to suffer when sharing a bottleneck with non-reactive UDP flows. In the second experiment, we study the protection of one TCP connection against a set of non-reactive over-transmitting UDP flows. Figure 9 shows the normalized throughput achieved by the TCP connection when it competes with $n-1$ UDP flows. Each UDP flow transmits at twice its fair share over the 10 seconds simulation time. In the figure, the y-axis represents the normalized throughput, which is defined as the throughput/fair-share ratio. Note that in the optimal situation, the normalized throughput should be equal to 1.0. The results in Figure 9 show that CSFQ performs slightly better than RFQ. As in [4], we also observe that DRR performs very well when there are 22 flows or less, and its performance deteriorates after that. This is because the TCP flow's throughput is significantly affected by the limited buffer share at the bottleneck when there are more than 22 flows. Both RFQ and CSFQ significantly outperform RED.

In the third experiment, we look at bursty traffic sources. We use a simplified MPEG video source model, which has a group of picture (GOP) pattern I-B-B-P-B-B-P-B-B. Each I frame is of size 10000 bytes, each P frame is of size 1500 bytes and each B frame has size 500 bytes[2]. In the simulation, each of the 20 video sources sends at 30 frames per second and the videos are carried by UDP packets each of size 500 bytes. Figure 10 shows the average throughput of each flow. In this case, both RFQ and DRR perform significantly better than CSFQ. We believe that the reason lies in the details of the fair share estimation. When there is a sudden change from uncongested to congested, the fair share estimation in CSFQ tends to produce large errors, To deal with this problem, the fair estimation is turned off (setting α to zero) when queue size is less than a threshold [19]. However, because video sources are approximately synchronized, the I frames from each source generate significant large bursts. The bursts from video sources sometimes get through when the fair share estimation is turned off, and cause unfair bandwidth sharing. In the case of RFQ, since the queue length is monitored and enforced around the *q_threshold*, the reaction to traffic bursts is quite fast.

[2]We use the deterministic frame sizes which have been reported as the mean values based on a study of MPEG encoded videos [20]. The deterministic frame sizes are unrealistic. However, it is sufficient to demonstrate the effect of video sources on congestion control algorithms.

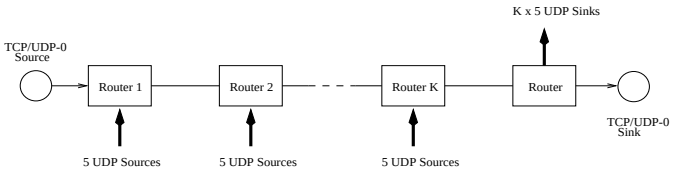


Fig. 11. Parking-lot network configuration. Each link is set at 10 Mbps with 1ms delay.

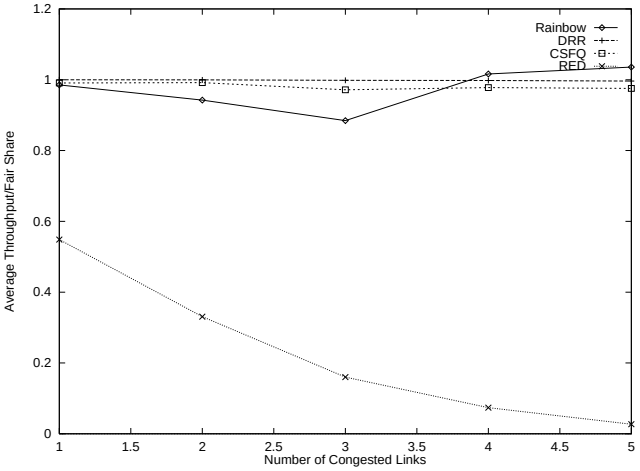


Fig. 12. Average throughput achieved by UDP-0 passing n consecutive links.

B. Multiple Congested Links

We also evaluated the performance of RFQ with multiple congested bottlenecks. Figure 11 shows the simulation topology, a typical multi-link parking-lot configuration. The number of congested links varies from 1 to 5. Each link's capacity is set at 10 Mbps with a propagation delay 1ms. Flow 0 traverses all of the congested links while other flows traverse fewer links.

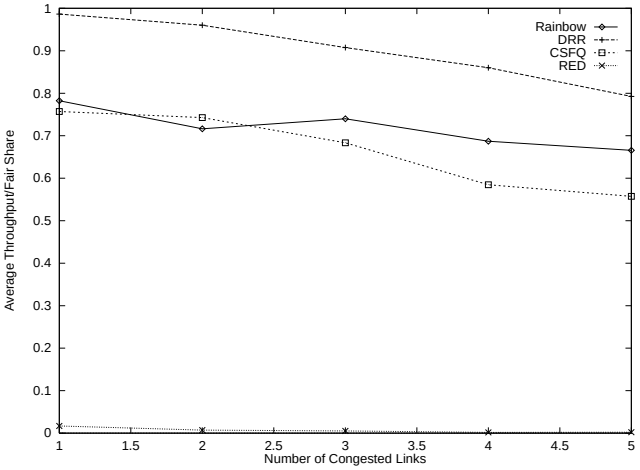
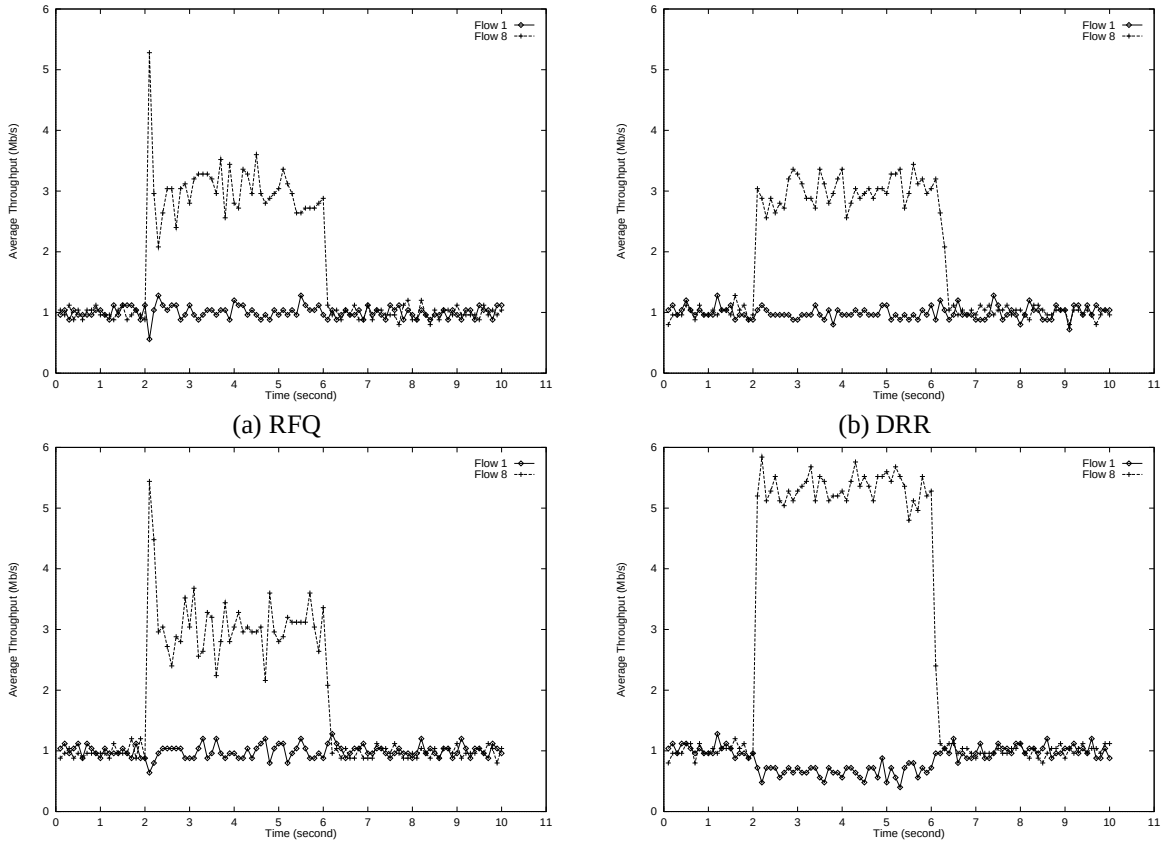


Fig. 13. Average throughput achieved by TCP-0 passing n consecutive links.



(a) RFQ (b) DRR

(c) CSFQ (d) RED

Fig. 14. Eight UDP flows initially sends at 1 Mbps. At time 2 second, flow 8 increases its rate to 8 Mbps.

At each of the routers 1 to K, five UDP sources are connected and they all terminate at the last router. In the experiments, all flows except flow 0 carry UDP traffic at an average of 4 Mbps, therefore, all links are congested.

In the first experiment, flow 0 is a UDP connection transmitting at 4 Mbps. Figure 12 shows the normalized bandwidth, i.e., the actual throughput divided by the fair share it should receive. DRR and CSFQ perform slightly better than RFQ. The performance of RED is significantly worse compared with the other three, and deteriorates as the number of congested links increases.

In the second experiment, we change flow 0 to be a TCP connection. Figure 13 shows the normalized bandwidth share it receives. The results show that the performance of RFQ is slightly better than CSFQ. In comparison, RED fails completely in protecting the TCP flow, and the TCP flow under RED receives very little bandwidth.

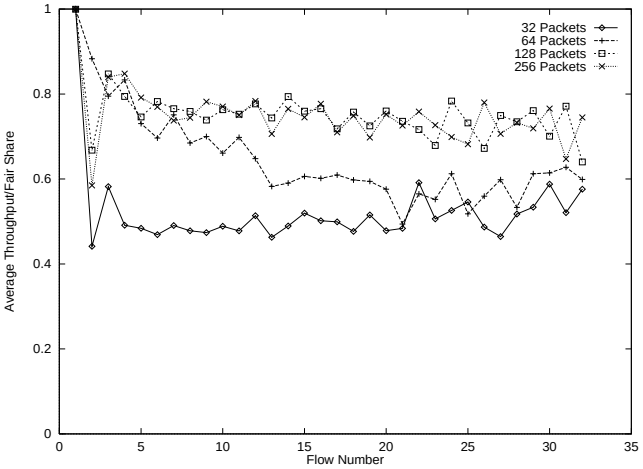


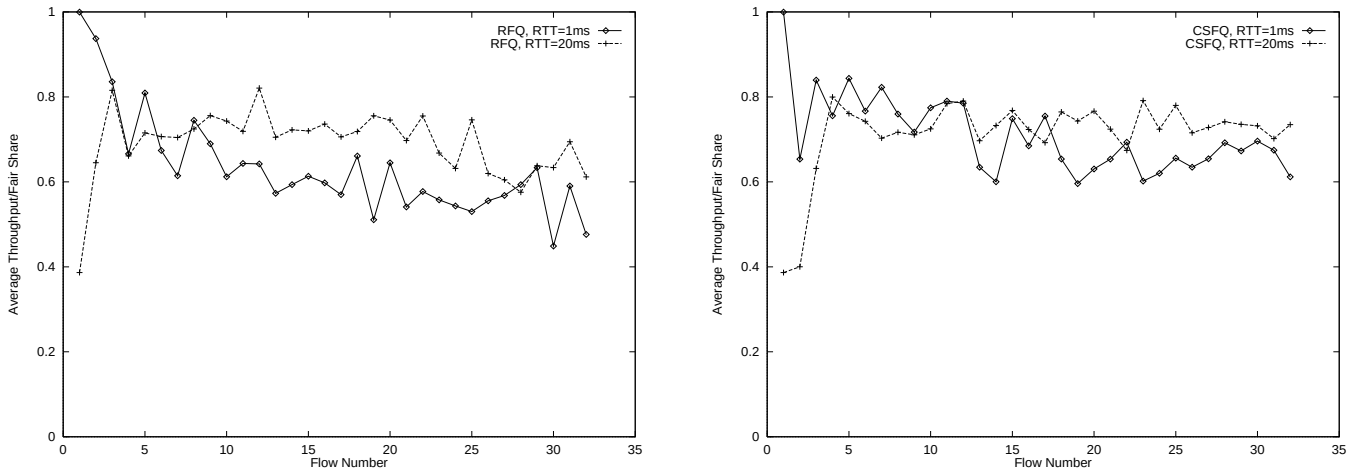
Fig. 15. Single link configuration with buffer size ranges from 32 packets to 256 packets at the bottleneck link.

C. Control Responsiveness

In this experiment, we investigate the control responsiveness of RFQ. At the beginning, each of the eight UDP flows sends at 1 Mbps to a single bottleneck link with capacity 10 Mbps, and the network is uncongested. After 2 second, flow 8 suddenly increases its rate to 8 Mbps. Under WFQ, all flows except flow 8 should still receive 1 Mbps throughput, while flow 8 consumes the rest of the link capacity which is 3 Mbps. After another four seconds, flow 8 reduces its transmission rate back to 1 Mbps. The average throughput of flow 1 and flow 8 are shown in Figure 14. The results show that both RFQ and CSFQ are able to quickly respond to transient load variations, adjust to and stabilize at the correct rates.

D. Performance Effects of Buffer Size

Since RFQ is an active queue management scheme, buffer size is always an important factor affecting the performance. For RFQ, buffer overflow essentially defeats the fair sharing of band-



(a) RFQ (b) CSFQ

Fig. 16. TCP performance when the link delay is 1ms and 20ms.

width because all incoming packets are forced to be discarded instead of being discarded according to the color threshold. In this section, we evaluate the effect of the buffer size on the performance of RFQ. Again, we use the single link topology and the link rate is set at 10 Mbps. One TCP connection competes with $n-1$ UDP flows each sending at twice the fair share. Figure 15 shows the normalized throughput of the TCP connections under different buffer sizes. The queue threshold *q_threshold* of RFQ is set at 60% of the buffer size in all experiments. The results in Figure 15 show that when the buffer size increases from 32 packets to 64 packets and to 128 packets, the throughput of the TCP connection improves. However, when the TCP flow competes only with one UDP flow, the throughput received by the TCP with a large buffer size (128 or 256 packets) is even lower than the throughput with smaller buffer size. That is because the long queue size increases the round trip delay for the TCP connection and it becomes the primary factor that limits the throughput of TCP. In our simulation, by decreasing the threshold, we reduce the RTT, but the threshold has to be set a level to limit false *congestion* signals which may cause the color threshold to fluctuate too quickly.

E. TCP Performance Under Various Round Trip Delay

As we have mentioned in the previous section, the RTT may affect the throughput of TCP connections. The speed at which TCP can react to packet dropping is a function of the RTT. In this section, we examine how different RTTs affect the fair share for a TCP connection. We use the single link topology with the bottleneck link rate set at 10 Mbps. A TCP connection competes with $n-1$ UDP flows. Each UDP flow sends at twice its fair share. We experimented with link delays of 1ms and 20ms. The normalized throughput is shown in Figure 16. The result indicates that longer RTT increases the amount of bandwidth the TCP receives. That is because the delayed acknowledgment postpones TCP's slow start when its packets are discarded. Therefore, the extended transmission period increases the connection's throughput.

F. Weighted RFQ

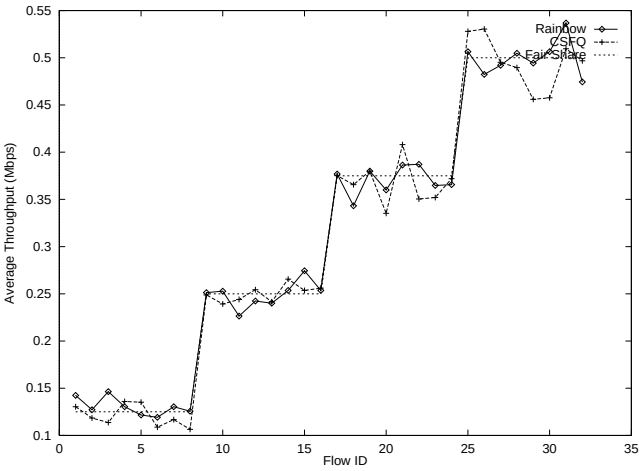


Fig. 17. Simulation of weight RFQ and CSFQ. Flows 1-8 have weight 1, flows 9-16 have weight 2, flows 17-24 have weight 3 and flows 25-32 have weight 4.

To evaluate how weighted RFQ performs, we repeat the simulation that 32 UDP flows share a single bottleneck link with 10 Mbps. We assign weight 1 to flow 1-8, weight 2 to flow 9-16, weight 3 to flow 17-24 and weight 4 to flow 25-32. Figure 17 shows the average throughput each flow receives in comparison with the fair share. The results show that both RFQ and CSFQ perform well in emulating weighted fair queueing.

V. Improving Performance of Layered Applications with RFQ

Resource heterogeneity and bandwidth constraints are likely to remain characteristics of the Internet. Over the years, layered multi-media applications have been proposed as a possible solution to these problems, as they offer great flexibility in delivering media to end users with performance best suited to individual characteristics. For example, in layered video delivery, the video stream is partitioned into a number of layers. The base layer can be played back with minimum quality, and each additional layer adds quality to the video. In most encoding schemes, the decoding of additional layers depends on the lower layers. Thus, the

lower layer data are more important than higher layer data.

This adaptation model works well in a heterogeneous Internet: a customer with constrained bandwidth can subscribe to a limited number of layers and achieve acceptable quality. If customers have enough bandwidth, they can subscribe to all layers to achieve the maximum quality. Hence, the quality received by end users is optimized without explicit adaptation by senders. The quality adaptation avoids the many round trip delays usually required by close-loop adaptation mechanisms, and it is well suited for real-time multimedia delivery and multicast applications.

Because of the prioritized layering approach in RFQ, the layered encoding in multimedia applications fits naturally with RFQ. With RFQ, layers in multimedia flows can be mapped directly to the colors by the edge routers. The more important data layers can be assigned lower color values while other data layers are assigned high color values. When congestion occurs inside the network, less important data packets are dropped first. There are at least two ways to implement the mapping at edge routers. Current Internet routers can already "look" into packets to determine the type of packets, and then they can assign colors based on the packet type. A more feasible solution is that the application at the end station may label its packets with layer information in packet headers, and the edge routers can then retrieve this information to assign colors.

We conducted simulations to evaluate the benefit of RFQ for layered video. Our model for video traffic is based on the MPEG compressed video framing and transmission scheme. MPEG uses three types of frames (I, P, and B frames) to encode data using relational and temporal compression. Intrapicture (I) frames provide periodic updates to the complete frame. Predicted picture (P) frames are coded relative to a past picture (I or P). Interpolated picture (B) frames are coded relative to a past and future picture. This encoding scheme achieves significant bit-rate reduction, but a loss of a frame may corrupt the frames that depend on it. Thus the *goodput* instead of the simple data throughput is a good measure of the network performance because the data throughput measurement includes the useless data that could not be decoded by receivers. For example, a P or B frame has to be discarded if it depends on a dropped I frame even if the P or B frame has been successfully received.

In our simulation, we use a simplified video source model. A video stream is encoded in a GOP pattern I-B-B-P-B-B-P-B-B. I frames have size of 10000 bytes, P frames have size of 1500 bytes and B frames have size of 500 bytes. The video is encoded at 30 frames per second, with an average rate of 426Kbps. In the network, the video is carried by packets with MTU of 500 bytes, and the loss of any single packet causes that frame to be undecodable. With RFQ, packets belong to I frames are labeled with lower colors than packets belong to P and B frames, and packets belong to P frames are labeled with lower colors than packets belong to B frames. However, the average rate of traffic in each color still obeys the rules described in Section III. In the simulation, we measure the goodput achieved by the video application. We define the goodput as the successful delivery of both the video frames and their reference frames.

In Figure 18, we show the goodput of a video source that shares a single bottleneck link with n UDP flows. The link has a capacity of 10Mbps. Each UDP sends at 100Kbps. We compared the performance of RFQ, CSFQ and DRR in the simulation. The results show that RFQ consistently outperforms DRR and CSFQ in goodput. It is interesting to note that DRR performs worse than CSFQ. This is because the dropping in DRR tends to spread out fairly evenly across the packets, which actually reduces the goodput.

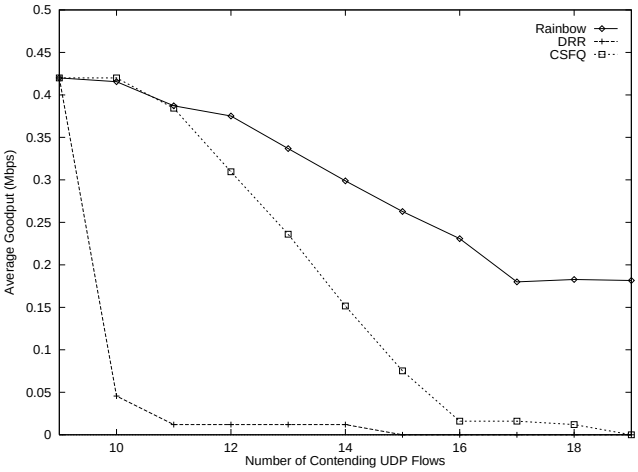


Fig. 18. An MPEG flow shares a bottleneck link with n UDP flows, which each sends at 1/10 of the link capacity 10Mbps.

VI. Concluding Remarks

In this paper, we presented Rainbow Fair Queueing (RFQ), a scheme for achieving approximate fair bandwidth sharing without per-flow state. RFQ achieves fair sharing by first dividing each flow into a set of layers at the edge of the network, and then marking each layer with a different color. Within the core of the network, routers maintain a color threshold C; packets with a color label greater than C are dropped. Because of the structure in colored layers, the discarding of packets is approximately fair.

We have evaluated RFQ together with CSFQ, DRR and RED with several different configurations and traffic sources. The simulation results show that RFQ is able to achieve approximately fair bandwidth sharing in all of these scenarios. The performance of RFQ is comparable to that of CSFQ, and it performs much better than RED. The simulations also show that RFQ outperforms CSFQ and DRR with respect to goodput when applications take advantage of using color layers to encode preferences.

The RFQ scheme further pushes the complexity away from the core towards the edges of the network. The core routers operate using simple FIFO scheduling with a threshold-based packet discarding mechanism. The core router algorithm only uses simple primitive operations which makes it amenable to hardware implementation. The algorithms for the rate estimation and color assignment at the edge routers, and the adjustment of threshold in the core routers, are still subjects for future research. However, our experiments with the current schemes have produced very encouraging results.

Although RFQ is examined in the context of fair bandwidth sharing for individual flows, it is possible to extend the scheme to aggregated flows that comprise many individual flows. In fact,

changes would only be needed at the edges of the network; the core router algorithm can remain the same. Instead of assigning colors to an individual flow, the edge routers simply bundle or classify packets into an aggregate flow (e.g., all packets from a source network to a web server), and color the aggregate flow the same way. One issue is that an aggregate flow may take multiple paths. For example, a source network may connect to two different ISPs and load balance between them. Thus, packets from the same source network to the same destination may follow two different paths. This will affect the structure of the color layers of the packets in both paths. However, we may be able to overcome the problem with a random color assignment. The details of such a scheme are also left to future work.

REFERENCES

[1] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *SIGCOMM Symposium on Communications Architectures and Protocols*, Sept. 1989.

[2] Abhay Parekh, *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks*, Ph.D. thesis, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Feb. 1992.

[3] Dong Lin and Robert Morris, "Dynamics of Random Early Detection," in *Proceedings of SIGCOMM'97*, Oct. 1997.

[4] Ion Stoica, Scott Shenker, and Hui Zhang, " Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks," in *Proceedings of SIGCOMM'98*, Sept. 1998.

[5] M. Shreedhar and George Varghese, "Efficient Fair Queuing using Deficit Round Robin," in *Proceedings of SIGCOMM'95*, Sept. 1995.

[6] Sally Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, Aug. 1993.

[7] Jon C.R. Bennett and Hui Zhang, "Hierarchical Packet Fair Queueing Algorithms," in *Proceedings of SIGCOMM'96*, Aug. 1996.

[8] S. Golestani, "A Self-Clocked Fair Queueing Scheme for Broadband Applications," in *Proceedings of IEEE INFOCOM'94*, June 1994.

[9] Steven McCanne, Van Jacobson, and Martin Vetterli, "Receiver-Driven Layered Multicast," in *Proceedings of SIGCOMM'96*, Sept. 1996.

[10] Lorenzo Vicisano, Luigi Rizzo, and Jon Crowcroft, "TCP-like Congestion Control for Layered Multicast Data Transfe," in *Proceedings of IEEE INFOCOM'98*, Mar. 1998.

[11] Xue Li, Sanjoy Paul, and Mostafa Ammar, "Multi-Session Rate Control for Layered Video Multicast," in *Proceedings of Symposium on Multimedia Computing and Networking (MMCN99)*, Jan. 1999.

[12] L. Wu, R. Sharma, and B. Smith, "Thin Streams: An Architecture for Multicasting Layered Video," in *NOSSDAV'97*, May 1997.

[13] Lorenzo Vicisano and Jon Crowcroft, "One to Many Reliable Bulk-Data Transfer in the MBone," in *Proceedings of the Third International Workshop on High Performance Protocol Architectures HIPPARCH'97*, June 1997.

[14] Michael Donahoo, Mostafa Ammar, and Ellen Zegura, "Multiple-Channel Multicast Scheduling for Scalable Bulk-Data Transport," in *Proceedings of Infocom'99*, Mar. 1999.

[15] Steven Blake and et al., "An Architecture for Differentiated Services," Dec. 1998.

[16] Van Jacobson, "Congestion Avoidance and Control," in *ACM SIGCOMM 88*, 1988.

[17] Juha Heinanen and Roch Guerin, "A Two Rate Three Color Marker," Internet Draft, `<draft-heinanen-diffserv-trtcm-01.txt>`, May 1999.

[18] UCB/LBNL/VINT, "Network Simulator - NS (Version 2)," `http://www-mash.cs.berkeley.edu/ns/`.

[19] Ion Stoica, "NS-2 Simulation for CSFQ," `http://www.cs.cmu.edu/~istoica/csfq`, 1998.

[20] Samrat Bhattacharjee and Martin McKinnon, "Performance of Application-Specific Buffering Schemes for Active Networks," Tech. Rep. GIT-CC-98-17, College of Computing, Georgia Tech, 1998.