Design Considerations for Supporting TCP with Per-flow Queueing

Bernhard Suter, T. V. Lakshman, Dimitrios Stiliadis, and Abhijit K. Choudhury

Bell Laboratories
Lucent Technologies
101 Crawfords Corner Road
Holmdel, NJ 07733, USA

{suter,lakshman,stiliadi,akc}@research.bell-labs.com

Abstract

In this paper, we investigate the extent to which fair queueing (and its variants), in conjunction with appropriately tailored buffer management schemes, can be used to achieve the following goals for TCP traffic: 1) alleviate the inherent unfairness of TCP towards connections with long round-trip times, 2) provide isolation when connections using different TCP versions share a bottleneck link, 3) provide protection from TCP-unfriendly traffic sources (which might include TCP ACKs since they are not loss-responsive) and misbehaving users, 4) alleviate the effects of ACK compression in the presence of two-way traffic, 5) prevent users experiencing ACK loss (which causes their traffic to be bursty) from significantly affecting other connections, 6) provide low latency to interactive connections which share a bottleneck with "greedy" connections without reducing overall link utilization. The paper proposes new buffer management schemes to be used in conjunction with fair queueing, so as to achieve the above goals for TCP, and compares the performance of the proposed schemes to the performance obtained using RED for packet dropping.

I. Introduction

In this paper we study the performance of TCP with per-flow queueing. The study is motivated by recent research in scheduling with per-flow queueing that has resulted in the development of efficient implementation methods (see [22] for a survey) for fair queueing. Consequently, it has become feasible to implement fair-queueing in switches and routers to provide delay bounds to some traffic classes and to achieve isolation from misbehaving sources. A primary reason for interest in fair queueing has been its ability to provide end-to-end delay bounds for leaky-bucket controlled sources. However, if fair-queueing systems are deployed in routers, it is important to examine the usefulness of fair-queueing (a term which we henceforth use loosely to denote some form of per-flow guaranteed bandwidth scheduling) for major traffic sources which are not leaky-bucket controlled but are feedback controlled and adaptive such as TCP.

We examine whether fair-queueing, in conjunction with appropriate buffer management schemes, can be used to achieve the following goals for TCP traffic:

Alleviate the inherent unfairness of TCP towards connections with long round-trip times : TCP's bias against connections with large round-trip delays and against connections traversing a large number of congested gateways is because these connections need larger windows to maintain the same throughput as shorter round-trip time connections. However, since window growth is clocked by returning acknowledgments (ACKs), large round-trip time connections which need large windows actually have their windows grow slower. This results in discrimination against long connections which varies between the inverse and the inverse square of the round-trip times (depending on the amount of queueing delays in the network) [5, 14].

We measure fairness by comparing the goodput of greedy sources after a long elapsed time. We define the *fairness coefficient* to be the coefficient of variation (standard-deviation over mean) of individual throughputs as a fraction of the total link capacity.

Provide isolation when connections using different TCP versions share a bottleneck link : It is known that TCP Reno is prone to phase effects and is not resilient to multiple losses from the same window [5]. However, TCP Reno works very well when losses are isolated. Depending on the nature of losses (which could depend on the nature of other traffic) one version of TCP may degrade the performance of another version.

Provide protection from TCP-unfriendly sources : Any source which, upon detection of loss, does not reduce its sending rate as drastically as TCP can be thought of as a TCP-unfriendly source. This is because if these sources and TCP sources share buffers and bandwidth in an uncontrolled manner then TCP sources will be "pushed out" and will achieve very little throughput. An example of a TCP unfriendly-source is TCP-ACKs itself since there is no adaptation to loss in the TCP reverse path. Also, UDP sources, which are not subject to the same adaptation as TCP, are possible TCP-unfriendly sources. We would like to protect TCP sources from TCP-unfriendly sources since the alternative approach of making every source TCP-friendly is not desirable, in many situations, since TCP's throughput drops off very rapidly with the bandwidth-delay product for a given loss rate [15].

Alleviate the effects of ACK compression in the presence of two-way traffic : ACK compression results from bunching of data packets which consequently causes bunching of ACK packets (which are queued behind data packets) in an FCFS scheduler. This bunching is preserved causing network traffic to be bursty [24]. Fair-queueing can serve ACKs separately and can eliminate ACK compression.

Provide low latency to interactive connections which share a bottleneck with "greedy" connections without reducing overall link utilization.

In the absence of fair queueing, the most widely used method for achieving some of the above goals is a packet dropping scheme called Random Early Detection (RED) [6]. While RED is a considerable improvement over tail-drop, the use of appropriate per-flow buffering and scheduling should allow us to improve upon the performance of RED. Merely using fair-queueing, without appropriate buffer management, does not lead to better performance. Consequently, in this paper we propose new buffer management schemes to be used with fair-queueing. We propose two per-flow dropping policies - Longest Queue Drop (LQD) and Dynamic Soft Partitioning with Random Dropping (RND) and compare these with the popular global dropping policy for TCP traffic, RED. We do this for both a global FCFS scheduler and a per-flow FQ scheduler (Frame Based Fair Queueing [21]). We study the performance of various combinations of scheduling (FCFS and FQ) and packet dropping schemes (RED, LQD and RND).

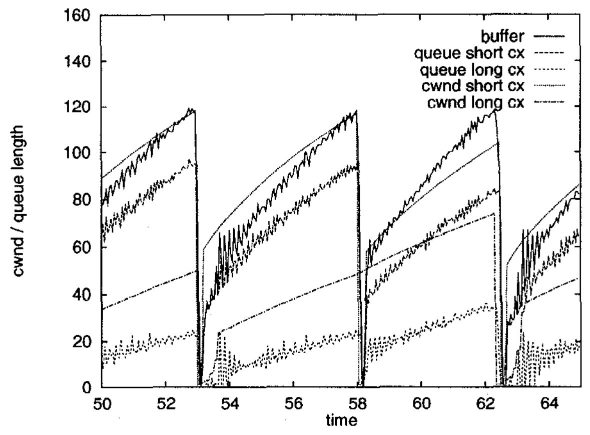


Fig. 1. Window and queue length evolution for two TCP Tahoe connections with RTT 20ms and 100ms sharing a 10Mbps bottleneck link with 100ms worth of buffer. Dropping policy is tail-drop with FCFS scheduler.

Our results indicate that fair queueing combined with the dropping policies that we propose can provide considerable gains over both FCFS-RED and FQ-RED. Furthermore, with the Internet becoming an important commercial infrastructure, it is necessary to provide flow isolation, and protection against malicious users. The schemes that we propose provide the required strong isolation amongst flows.

II. Buffer Management Schemes

Since TCP reacts to loss by reducing its window size, we expect TCP behavior not only to be influenced by the scheduling policy but also by the packet dropping policy. Per-flow scheduling can allocate bandwidth fairly among backlogged connections and can provide low queueing delay for low rate connections. However, for TCP, fair bandwidth allocation does not result in fair bandwidth usage. Only per flow queue management and dropping can provide fair usage, isolation, and protection from malicious or pathological behavior created by other flows.

Previous studies with FCFS scheduling have shown how TCP performance can be improved just by employing a different dropping strategy [6, 16]. As per-flow scheduling requires packets belonging to different connections to be identified, this valuable information should be used to optimize decisions regarding which packet to drop when congestion occurs.

A. Global Schemes

As global schemes are not able to distinguish packets from different connections, dropping decisions have to be made globally for a whole aggregate packet stream. We first consider the most simple policy: tail-drop (TD), where any newly arriving packet is rejected when the buffer is full. Figure 1 shows the evolution of congestion window and buffer state for two TCP connections with a ratio in round–trip times of about 5 sharing a common bottleneck link using FCFS scheduling and a tail-drop buffer. This system discriminates against connections with long round-trip times.

1. Since the window growth rate of a TCP connection depends on its round–trip time by $\frac{dW}{dt} = \frac{W}{\text{RTT}}$ in slow-start and $\frac{dW}{dt} = \frac{1}{\text{RTT}}$ in congestion avoidance mode, the window and thus the throughput of the short connection increases faster.
2. In FCFS scheduling the service given to different backlogged connections is roughly proportional to their share of the buffer. The short connection, being able to achieve a backlog faster, gets more service and achieves a higher window growth rate leading to an even faster increase in the buffer share.
3. When the buffer fills up, all connections are very likely to lose packets independently of their share of the buffer.

In this example with FCFS scheduling, the relative throughputs for the short and long connections are 0.84 and 0.13 respectively. Replacing FCFS with a Fair Queueing (FQ) scheduler reduces the unfairness due to (2) but cannot prevent unfairness resulting from (3). Even when not considering fairness, any dropping policy with property (3) is unable to provide flow isolation and protection from TCP-unfriendly sources. Amongst global dropping policies, a big improvement on tail-drop is Random Early Detection (RED) [6]. RED randomly drops incoming packets with a probability which is an increasing function of the average queue size. The idea is to not allow the full buffer to be used and instead limit average queue lengths to some fraction of the available buffer capacity. The empty buffer space can be used to absorb bursts. Also, the use of random dropping prevents synchronized window drops which can cause poor link utilization.

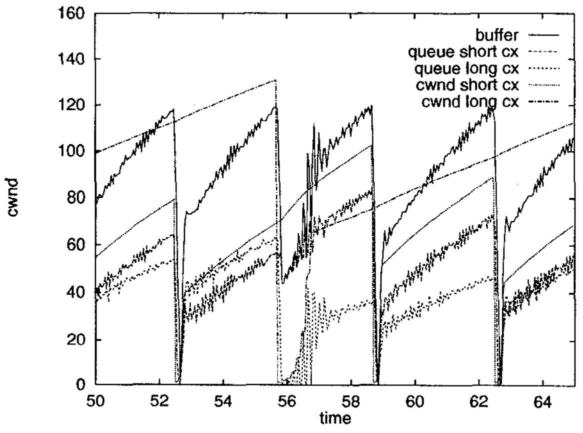


Fig. 2. Window and queue length evolution for two TCP Tahoe connections with RTT 20ms and 100ms sharing a 10Mbps bottleneck link with 100ms worth of buffer. Dropping policy is longest queue drop with FCFS scheduler.

B. Per-flow Schemes

If buffers could be statically partitioned, fair sharing of bandwidth could be achieved by partitioning the buffer space at the bottleneck link such that each flow has the same "normalized" buffer [14], where the normalized buffer of flow i is defined as the buffer allocated to flow i divided by its bandwidth-delay product. However, this requires the bottleneck router to have knowledge of the possibly time-varying round-trip times of each connection. In the absence of such information, we have to use a dynamic buffer sharing mechanism combined with dropping mechanisms that make TCP bandwidth usage as fair as possible.

We start with a global buffer pool of size B. Each connection i has a nominal buffer allocation b_i which can be thought of as connection i's guaranteed buffer size. In the absence of any a priori information, we can set $b_i = B/n$ for all i, where n is the number of backlogged connections. The b_is induce a soft partitioning of B. When connection i needs more than b_i buffers, it is allocated space from the available pool, provided of course that the total occupancy is less than B. If the total occupancy is B, and a connection j whose current occupancy q_j is less than b_j needs a buffer then we *push-out* the front packet from some connection whose current occupancy exceeds its allocation. We propose two methods for choosing the connection from which the push-out is done: (1) choose the connection i such that $(q_i - b_i)$ is the largest over all connections, or (2) select a connection at random from amongst the connections for whom $q_i > b_i$. We refer to the two schemes as LQD and RND respectively. Having chosen a connection, we drop packets from the front because this triggers TCP's fast retransmit/recovery feature faster and hence increases throughput [16].

B.1 Longest Queue Drop – LQD

The motivation for longest queue drop is that if connections are given equal weights, then connections which use the link more (get a higher share of the bandwidth unused by other connections) tend to have longer queues. Hence, biasing the packet drops such that connections with longer queues have higher drop rates should make the bandwidths sharing more fair.

Figure 2 shows the window evolution for the same system as that studied in 1 except that tail drop is replaced by Longest Queue Drop. The relative throughputs for the short and the long connections are 0.645 and 0.341 with FCFS, and 0.52 to 0.47 with FQ. FCFS with LQD actually achieves better fairness than FQ with tail-drop showing the importance of the packet dropping strategy. Fair Queueing with tail drop only achieves relative throughputs of 0.61 and 0.37. Clearly, FQ with LQD works very well.

In addition, the Longest Queue Drop policy offers some flow isolation and protection since if one connection misbehaves consistently, only this connection experiences an increased loss rate. Fair Queueing with Longest Queue Drop guarantees zero loss and low queueing delay to connections which do not exceed their bandwidth share of $\frac{C}{n}$ and this is independent of the behavior of other connections.

B.2 Dynamic Soft Partitioning with Random Drop – RND

Longest Queue Drop may lead to excessive bursty losses if in a system with many connections one queue is considerably longer than the second longest. TCP Reno type implementations are known to behave badly in presence of bursty loss. Therefore we propose a slight modification of the above scheme to reduce the amount of bursty loss. The backlogged connections are grouped into two subsets : those with occupancy q_i greater than b_i, and those with $q_i \leq b_i$. >From the set of queues above their allocation, one is picked randomly and a packet is dropped from the front. The flow is then put back into the set from which random samples are picked. This might cause the same flow to be picked more than once. The reason we pick a random connection is that we want to avert the possibility that all flows lose a packet during a congestion episode because this will cause synchronized window drops and loss of throughput when the buffer goes empty (due to all flows simultaneously reducing their windows by half or to 1).

B.3 Approximated Longest Queue Drop - ALQD

Longest Queue First dropping requires searching through the backlogged queues in order to determine which is the longest queue. We propose a variant of LQD that is particularly easy to implement, Approximated Longest Queue Drop (ALQD). A register holds the length and identity of the longest queue as determined at the previous queueing operation (queue, dequeue, drop). On every queueing event, the current queue length is compared with the longest queue identified in the register. If it is the same queue, the queue length in the register is adjusted. If the current queue is longer, its identity and length are now stored in the register. A similar scheme called Quasi-Pushout has been proposed recently in [17].

ALQD requires only $O(1)$ complexity in time and space. However, its state does not reflect exactly the state of the system. So optimal behavior at all times cannot be ensured especially when scheduling weights vary over a very wide range. A scenario could be constructed where ALQD cannot free enough memory and some incoming packets would have to be dropped, thereby breaching temporarily the strict flow isolation property of Longest Queue Drop.

III. Performance

We use simulations to compare the performance of our proposed schemes to that of FCFS-RED and FQ-RED. The comparisons are done using a mix of TCP Tahoe and TCP Reno sources, bursty and greedy sources, one-way and two-way traffic, sources with reverse path congestion, and with widely differing round trip times. The system that we simulate is as shown in Figure 3. Sources have high-speed access paths to a router which is our sole bottleneck. The access path delays are set over a wide range to model different round trip times. The destinations are assumed to ack every packet. For one way traffic, we send ACKs over a non-congested path. For two-way traffic, the return path is through the router and there may be queueing delays. In particular, when the router uses FCFS scheduling, ACKs and data packets are mixed in the queues. With fair queueing, ACKs are handled as separate flows. For asymmetric traffic, we reduce the bandwidth of the return link from the destination to the router so that there is considerable reverse path congestion and ACK loss. The simulation results were obtained using a modified version of the **ns v1.1** network simulator [18]. The RED model is packet oriented and uses 25% of the buffer size as the minimum threshold and 75% as maximum threshold, and the queue weight is 0.002.

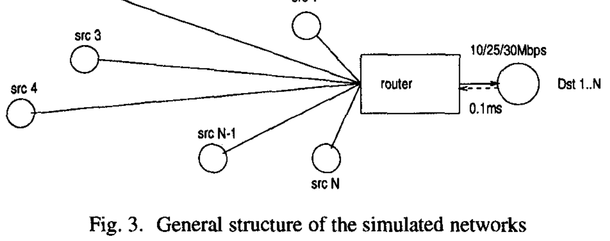


Fig. 3. General structure of the simulated networks

A. *Is fair queueing sufficient to provide fairness?*

Providing bandwidth guarantees alone to connections is not sufficient to improve fairness. Without an appropriate drop strategy, long round trip time connections are not able to use their share of the guaranteed bandwidth since guaranteed bandwidth is useful only if a flow has a backlog. For adaptive flows like TCP, the influence of the buffer management and dropping policy on throughput and fairness is significantly higher than the effect of scheduling because dropping policies significantly determine how quickly flows can build backlogs. In fact, FCFS scheduling with appropriate drop strategies can in many scenarios considerably outperform FQ.

We show below examples where LQD, RND and ALQD (per-flow drop strategies) perform significantly better than RED (a global drop strategy) for both FCFS and (even) for per-flow FQ schedulers.

A.1 Flow Isolation and protection from aggressive flows

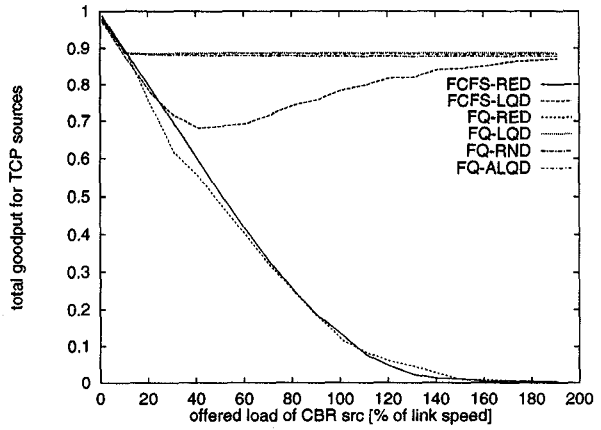


Fig. 4. Aggregate throughput for 10 TCP sources (2 to 160ms RTT) sharing a link with a loss-insensitive constant bit-rate source.

Figure 4 shows the fraction of the link bandwidth used by 10 TCP sources sharing a link with a non-responsive source. The sending rate of this source is varied from zero to twice the link bandwidth. As can be seen from the figure, in this scenario, with a consistent overload produced by a TCP-unfriendly source, the buffer management and dropping strategy have a stronger impact on bandwidth sharing than the scheduling policy itself.

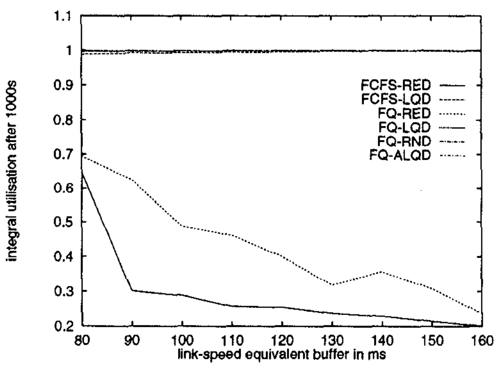
When the constant bit rate source increases its rate, for both FCFS and fair queueing with global RED there is very similar TCP-source performance degradation. However, with LQD, ALQD, or RND drop policies, an FQ scheduler is able to almost perfectly maintain the guaranteed rate of $\frac{1}{11}$ per flow, irrespective of whether the flow is TCP-like or is totally loss insensitive.

Even with FCFS, an LQD dropping strategy is able to protect TCP sources by concentrating losses to the sources with the highest arrival rate. TCP sources do not suffer excessive loss as long as they keep their arrival rate below their guaranteed link share. Nevertheless, being greedier than the average flow is still advantageous since FCFS gives to each flow a share of bandwidth proportional to the fraction of the queue that this flow occupies.

A.2 Comparisons with slow reverse path

To avoid the previous scenario, one possibility is to separate TCP-unfriendly and TCP-friendly sources using static classification and then use class-based queueing. In this section, we show that this is not sufficient because even TCP-sources can be TCP-unfriendly. This is because the reverse-path of TCP has no congestion control. If there is sufficient bandwidth asymmetry between the forward and reverse paths, the reverse path can be congested by ACKs alone. Since ACKs are not subject to flow control, they can significantly degrade performance for any other TCP flows sending data along this congested path (for a detailed explanation and analysis, see [15]). Of course, another case where static classification is insufficient is when there are incorrectly implemented TCP sources which send packets more aggressively than they are permitted.

a)



b)

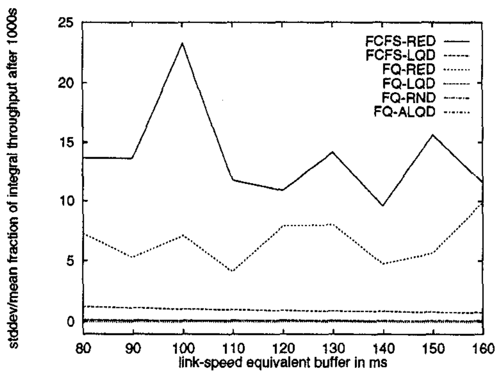


Fig. 5. Throughput (a) and fairness coefficient (b) as a function of buffer size with 20 connections over an asymmetric bottleneck link with 10Mbps/100 Kbps capacity(TCP Tahoe and Reno 20ms - 160ms RTT)

We examine TCP with a slow reverse path in some detail because of the following reasons: (1) This is an aspect which has not been studied well with regard to TCP performance with RED dropping, (2) recent studies [19] show that more than 50% of the measured routes in the Internet are asymmetric and follow different paths (and hence may experience reverse path congestion even when the forward path is congestion free), and (3) asymmetric bandwidth access networks such as ADSL severely reduce the reverse channel availability for ACKs in the presence of bidirectional traffic. A recent performance study [15] of TCP with drop-tail and drop-front queueing shows that TCP behavior is very bursty in the presence of reverse channel congestion and that the mode of operation is quite different from that of the forward path congested case. Also, fast retransmit/recovery does not work as intended.

TCP ACKs are generated purely as a response to data packets, regardless of the congestion state of the reverse path. Also, TCP

ACKs are cumulative. Therefore, loss of ACKs results only in increased source burstiness as long as the last ACK sent is not lost. From [15], we know that with a global dropping strategy, flows can be locked out when they traverse a link which is persistently congested by ACKs from other connections. This is due to high loss rate for data packets, and due to losses of ACKs corresponding to packets sent in the slow start phase. This is the main reason for the tremendous unfairness of the global dropping schemes (RED in this example) in Figure 5.

Besides the lock-out of connections, the reduced throughput of RED is due to the high number of timeouts that result from either retransmitted packets being dropped or the ACKs corresponding to retransmitted packets being dropped. In [15] it was shown, that when the available buffering in the forward direction is greater than the asymmetry (the ratio of data packet transmission time in the forward path to ACK packet transmission time in the reverse path), every TCP cycle ends in a time-out and hence greatly reduces throughput. Also, it was shown that this is eliminated by using a drop-from-front strategy, that results in the preservation of the latest ACK, in the reverse buffer. Since RED drops only incoming packets, it does not have this preserve last-ACK property of drop-from-front. Consequently, simulations show that RED in an asymmetric system results in a very large number of time-outs (as predicted by the analysis in [15]). The schemes FQ-RND, FQ-LQD, FQ-ALQD, and FCFS-LQD use drop-from-front in all buffers. Hence, they are not prone to ending each cycle with a time-out (for a detailed explanation of this, see [15]).

With the above background, the throughput and fairness plots (Fig. 5) can be explained as follows: Both RED policies have poorer throughput because the ACKs corresponding to retransmitted packets are lost 75% of the time for our asymmetry value of four (note that this is the bandwidth asymmetry divided by the ratio of packet size to ACK size). This results in a timeout in at least 75% of TCP cycles greatly reducing throughput. Other time-outs happen because of multiple losses in the forward path and losses of retransmitted packets in the forward path. On the other hand, drop from front in the reverse path eliminates these time-outs almost completely. Since time-outs are expensive, both RED schemes have poorer throughput than the other schemes including FCFS-LQD. When the reverse path uses a global dropping policy, it has been shown in [15] that the buffer occupancy tends to be dominated by a few flows causing lock out of other flows. FQ-RED has no mechanism to counteract this. Hence, the fairness of FQ-RED is very poor. However, FCFS-LQD prevents lockout since it drops packets from flows that dominate the buffer. Hence, its fairness is better than that of FQ-RED.

Both FQ-RND and FQ-LQD/FQ-ALQD work very well because they combine the advantages of per-flow queueing with the time-out elimination of drop-from-front. FQ-LQD has the further advantage in that it has a built-in bias against dropping retransmitted packets. This is because when the source detects a loss by receipt of the first duplicate ACK it stops sending packets. The retransmitted packet is sent only after the third duplicate ACK is received. During the intervening interval when the source is forced by TCP to be silent, the queue corresponding to the flow is drained at least at its minimum guaranteed rate and therefore it is less likely to be the longest queue when the retransmitted packet arrives. Hence, the inherent bias against dropping retransmitted packets. Though this bias is not limited to asymmetric networks, the bias is enhanced in asymmetric networks due to the slow reverse channel dilating, by the asymmetry factor, the interval between receipt of the first and third duplicate ACKs. Since loss of retransmitted packets causes an expensive time-out, this bias improves the performance of FQ-LQD. FQ-RND has this bias as well, though to a lesser degree. The reasoning is somewhat similar to that for FQ-LQD: during the interval between receipt of the first and third duplicate ACKs, the flow's queue drains at a rate equal to at least its guaranteed rate (since the source is silent) and the queue occupancy could fall below the reservation parameter for that flow. In that case, when the retransmitted packet arrives, it is not dropped even if the aggregate buffer is full. With these advantages, FQ-LQD, FQ-ALQD and FQ-RND have the best performance as is clearly evident in Figure 5.

B. One-way mixed greedy sources

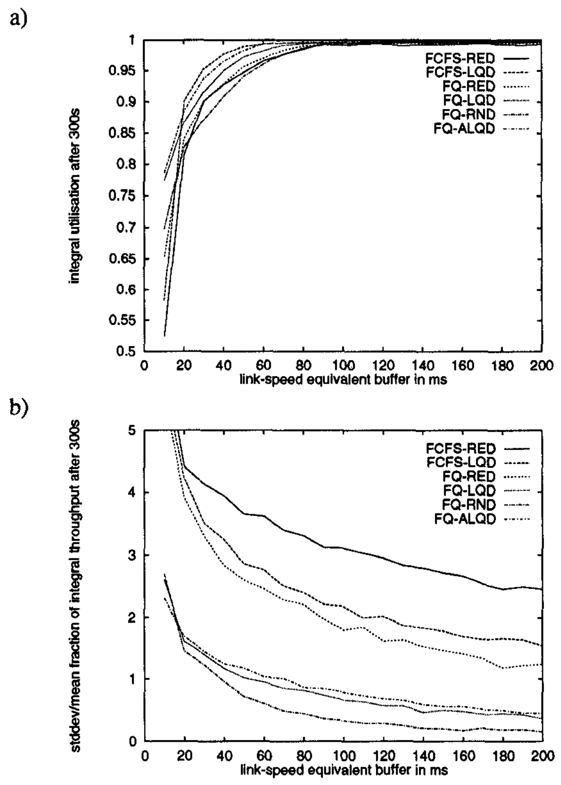


Fig. 6. Throughput (a) and fairness coefficient (b) as a function of buffer size for 40 TCP connections (Tahoe and Reno) sharing a 30Mbps bottleneck link. Round trip times vary from 120ms to 200ms for (a) and from 20ms to 200ms for (b).

To study how different schemes perform when there are multiple TCP implementations with differences in round trip times as well, we simulated 40 one-way TCP Tahoe and Reno connections with widely different round trip times. Results are shown in Figure 6. When the buffer size is approximately equal to or greater than the bandwidth-delay product (delay being that corresponding to short round-trip time connections), the throughputs are all very close to each other and very high. This is not much of a differentiating factor unless buffer sizes are smaller. FQ-LQD and FQ-RND still have the higher throughputs. Note also, that the fairness of FQ-LQD and FQ-RND are much higher than the FCFS schemes including FCFS-RED. With a mix of round-trip times, the buffer is mostly occupied by short round-trip time flows. When the buffer gets full, the FQ-LQD policy picks one of these short connections for packet drops and hence reduce their throughput advantage. With FQ-RED on the other hand, flows with short round-trip times get the majority of packet drops. However, since incoming packets are picked for dropping some long round-trip time connections may get dropped as well even though their queue occupancy is negligible. Hence, it does not perform as well as FQ-LQD.

FQ-RND first selects flows whose occupancies are above their reservation parameters. These are likely to be the flows with short round-trip times. With only a moderate number of these, FQ-RND tends to be like FQ-LQD. However, FQ-RND spreads the losses over several flows unlike FQ-LQD which could drop several packets at a time from the same flow. Multiple losses from a flow can adversely affect TCP Reno. Hence, the fairness coefficient of FQ-RND is better than that of FQ-LQD. With a large number of flows, FQ-RND will spread losses more evenly amongst flows than FQ-LQD. So its performance with a mixture of sources should continue to be better than that of FQ-LQD as the number of sources increases. Simulations in Figure 7 validate the above argument. However, either FQ-LQD or FQ-RND can be used since the differences are not high enough for one of them to be preferred. As expected the simplified FQ-ALQD performs somewhat worse than FQ-LQD.

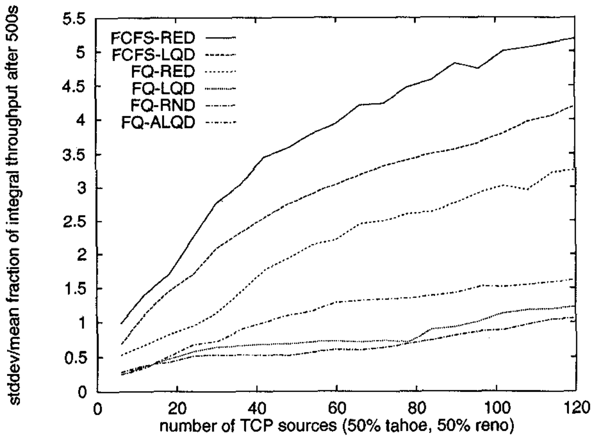


Fig. 7. Fairness coefficient as a function of the number of connections over a bottleneck link of 25Mbps and 100ms worth of buffer. TCP connections are in equal numbers Tahoe and Reno implementations, with 20ms, 80ms and 160ms RTT.

C. Two-way traffic

We also studied the performance of the above mix of sources with two way traffic. In the FCFS case, ACK and data packets mix in the same buffer, and traffic may be bursty due to ACK compression [24]. With fair queueing, the ACKs can be treated as separate flows. ACK queueing can happen but sustained congestion is prevented unlike in the asymmetric case. Figure 8 shows that again FQ-LQD and FQ-RND perform the best both in fairness and throughput. The reasons are the same as in the case of one-way traffic. The throughput of FQ-RND is lower than that of FQ-LQD. This is probably because FQ-RND, with only a few (about 4 in this case) short round trip time flows, occasionally picks all of them for packet drops and hence synchronizes them causing occasional aggregate throughput loss. FQ-LQD picks only one flow at a time and hence does not synchronize them. Also, TCP Tahoe throughput will be higher under bursty loss caused by FQ-LQD than with random losses (for the same loss rate).

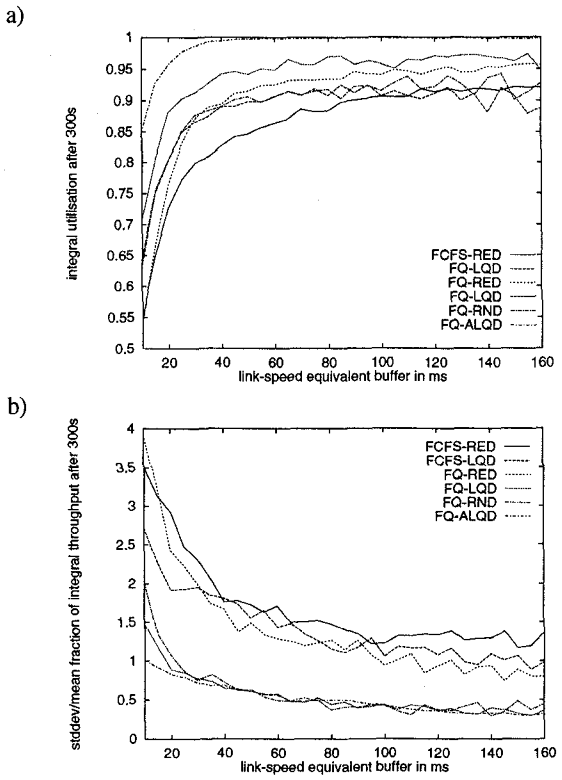


Fig. 8. Throughput (a) and fairness coefficient (b) for two–way traffic as a function of buffer size with 20 TCP connections in each direction. (TCP Tahoe and Reno with 40ms, 50ms, 56ms, 100ms, 110ms, 120ms, 140ms, 150ms, 156ms, and 160ms RTTs, 45Mbps bottleneck link capacity)

D. Mixture of bursty and greedy sources

To test the robustness of our schemes, we studied the throughput and fairness measured across greedy TCP sources in the presence of bursty ON-OFF TCP sources. To model the high variability of web transactions, we used Pareto sources with the complementary distribution function given by:

$$P[\text{size} > x] = \left(\frac{3x}{200 \times 10^3}\right)^{-1.5}, \; x > \frac{200 \times 10^3}{3}. \quad (1)$$

Note that for these parameters the coefficient of variation is infinite and so the aggregate traffic has long range dependence [13]. Mean burst size is 15 packets and mean waiting time after complete transmission of a burst is 1s.

The fairness coefficients are shown in Figure 9. Once again, FQ-RND and FQ-LQD have the best fairness coefficients. FQ-RND is better than FQ-LQD because FQ-RND is less likely to pick a Pareto source when choosing a flow to drop packets from.

E. Insufficient flow space

An argument against per-flow queueing is that it may not scale to the numbers that are required in a backbone router. A prototype implementation capable of handling 1 million packets per second and 64K queues was built with inexpensive FPGA's, and our experience indicates that such a design is certainly feasible. Furthermore, with a reasonably large number of queues, a much higher number of flows can be supported since fair queueing requires flow-state to

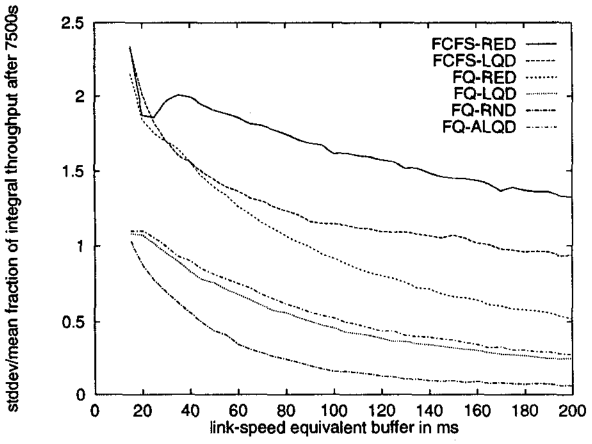


Fig. 9. Fairness coefficient as a function of buffer size for 20 greedy TCP sources (TCP Tahoe and Reno with 40ms, 50ms, 56ms, 100ms, 110ms, 120ms, 140ms, 150ms, 156ms, and 160ms RTTs) sharing a 25Mbps bottleneck link with a similar set of 20 pareto on/off sources.

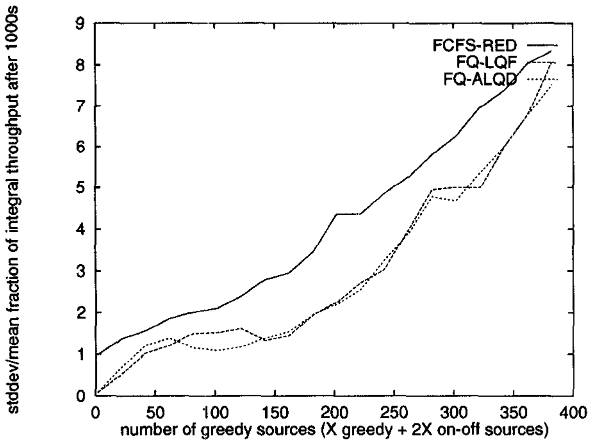


Fig. 10. Fairness coefficient as a function of active flows from greedy and on-off TCP sources (TCP Tahoe and Reno with 10 to 160ms RTTs) sharing a 10Mbps bottleneck link. Flows are replaced in a least recently used fashion (FQ-LQD) or randomly (FQ-ALQD).

be maintained only for backlogged flows and flows that have sent packets during a small window of time prior to the current instant [20, 21]. Also, there is a reasonable upper limit on the number of simultaneously active TCP connections that a link can support. If on a specific path there are more active connections than the equivalent of the delay-bandwidth product in packets (plus available buffer capacity), the average TCP window size is effectively less than 1 and throughput would be extremely low. On an OC-3 link, assuming 300ms round trip times and average packet sizes of around 500 bytes, the limit of one packet simultaneously per flow gives us only around 10 to 20 thousand flows.

Nevertheless, since flow space is finite, there is a possibility that flows may be replaced (from their initial queue assignment) during their lifetime. Hence, we simulated a situation where the number of active flows vastly exceeds the available flow space. Figure 10 shows the fairness index for a number of greedy TCP sources mixed with twice as many Pareto distributed on-off sources (mean burst 15 packets, mean idle time 10s) that generate a new flow every time they become active.

The simulated per-flow queueing system can support up to 128 simultaneous flows and so the number of simulated flows exceeds the number of queues. If all available queues are assigned to flows and new flows are detected, the state for some existing flows has to be replaced with the new ones. The FQ-LQD system uses a Least-Recently-Used (LRU) strategy to replace flow to queue assignments. As LRU can become expensive to implement, the simple ALQD scheme uses random replacement, which means that a newly detected flow randomly replaces any existing flow in the flow-to-queue mapping. The per-flow queueing systems still perform better than the FCFS-RED. Fairness and flow isolation is preserved to some extent.

IV. Implementation Issues

An accurate implementation of fair queueing and per-flow buffer management schemes results in $O(logN)$ complexity where N is the number of connections sharing the outgoing link. In addition, there is an $O(N)$ space requirement. Several variations of fair-queueing schemes have been proposed lately that reduce its complexity by discretizing the time-stamps or the available rates [21]. A variation of the per-flow buffering schemes which will allow an efficient implementation can be similarly derived.

Queues are classified into discrete bins based on their lengths, and with an exponentially increasing spacing between bins. Whenever a queue is modified (enqueue, dequeue or drop), it is moved to the appropriate bin (which is either the same, one above or below the current bin). The system keeps track of the highest occupied bin. When the buffer is full, any queue from the highest occupied bin is selected and its first packet dropped. If the buffer is measured in bytes, this operation may have to be repeated until enough space has been freed to accommodate the newly arrived packet due to variable packet sizes. To achieve true LQD, the queues in the highest occupied bin would either have to be maintained as a sorted list or searched for the longest queue every time. However, we have shown that the performance of the simpler ALQD scheme is close to that of LQD. Per-flow queueing clearly requires more memory than FCFS. But this memory cost is not much compared to the benefits of isolation, better performance, administrative flexibility, that per-flow queueing provides.

V. Summary of Results and Concluding Remarks

It is clear that fair queueing with per-flow buffering can provide benefits such as isolation and quality-of-service guarantees. It was not clear whether these benefits which have been shown for non-adaptive shaped flows would apply for adaptive flows such as TCP flows. We showed that fair queueing can benefit adaptive flows provided buffer management and packet dropping is done appropriately. We proposed schemes for TCP traffic that perform very well in conjunction with fair queueing and are much superior to FCFS. If the simplicity of FCFS is to be traded for the higher stability and fairness of fair queueing (which due to advances in hardware implementations techniques is feasible even in backbone routers), then there is little reason to persist with using a global dropping strategy. The best choices are to use TCP-aware buffering and dropping strategies such as FQ-LQD and FQ-RND. For very little additional complexity, these schemes provide good fairness and provide nearly perfect flow isolation and protection. Our results are summarized below:

1. Merely using a fair-queueing scheduler is not sufficient to improve performance and isolation. We illustrated this by showing that in situations where strong isolation is required,

like in the presence of misbehaving users or in asymmetric systems, FCFS-LQD gives much better performance that FQ-RED.

2. With a slow reverse channel and greedy sources, FQ-LQD and FQ-RND are by far the best choices. They have considerably higher throughputs than FCFS-RED and FQ-RED, and much higher fairness coefficients as well. RED has a disadvantage over push-out, in that an undue number of time-outs result, which it appears cannot be remedied without some form of drop-from-front queueing.
3. FQ-LQD has the best overall throughput for a mix of Tahoe and Reno connections with widely different round-trip times and over a wide range of buffer sizes. In addition, it has the best fairness metric when the number of short round-trip times is moderate. For a large number of connections with short round trip times, the fairness coefficient of FQ-RND is slightly lower than that of FQ-LQD. The throughput of both FCFS-RED and FQ-RED are generally lower than that of both FQ-RND and FQ-LQD. Their fairness coefficients are significantly lower.
4. In the same scenario as above but with two-way traffic, FQ-LQD again gives the highest throughput with FQ-RND being next. FCFS-RED is always lower in throughput than FQ-LQD. The fairness coefficient is the best again for FQ-LQD followed by FQ-RND and both have fairness coefficients much better than both FCFS-RED and FQ-RED.
5. For a mixture of bursty and greedy sources, FQ-RND has the best fairness coefficient followed closely by FQ-LQD. FQ-RED and in particular FCFS-RED have much poorer fairness coefficients.
6. The approximated implementation of LQD (ALQD), which does not require sorting or searching of queue lengths, has slightly worse performance than the ideal LQD but the degradation is such that it makes the complexity-performance tradeoff worthwhile.

REFERENCES

[1] J.C.R. Bennett and H. Zhang, "WF^{2}Q: Worst-case Fair Weighted Fair Queueing", *Proc. of INFOCOM 96*, pp. 120-128, March 1996.

[2] B. Braden, D. Clark, S. Shenker, "Integrated Services in the Internet Architecture: an Overview", RFC 1633, June 1994

[3] B. Braden, D. Clark, J. Crowcroft, B. Davie, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", Internet draft `draft-irtf-e2e-queue-mgt-recs.ps`, March 97

[4] A. Demers and S. Keshav and S. Shenker, "Analysis and simulation of a fair queueing algorithm", *Internetworking: Research and Experience*, pp. 3-26, 1990.

[5] S. Floyd and V. Jacobson, "On Traffic Phase Effects in Packet-Switched Gateways," *Internetworking: Research and Experience*, vol.3, no.3, pp.115-156, September 1992. (An earlier version of this paper appeared in *Computer Communication Review*, vol. 21, no. 2, April 1991.)

[6] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", *IEEE/ACM Transactions on Networking*, August '93.

[7] S. Floyd and V. Jacobson, "Link-sharing and Resource Management Models for Packet Networks", *IEEE/ACM Transactions on Networking*, August '95.

[8] L. Georgiadis, I. Cidon, R. Guérin, and A. Khamisy, "Optimal Buffer Sharing," *IEEE J. Select. Areas Commun.*, vol. 13, pp. 1229–1240, Sept. 1995.

[9] S.J. Golestani, "A Self-Clocked Fair Queueing Scheme for Broadband Applications", *Proc. of IEEE INFOCOM '94*, pp. 636-646, April 1994.

[10] P. Goyal, H.M. Vin and H. Chen, "Start-time Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks", *Proc. ACM SIGCOMM'96*, pp. 157-169.

[11] V. Jacobson, "Congestion avoidance and control," *Proc. ACM SIGCOMM '88*, pp. 314-329.

[12] V. Jacobson, "Berkeley TCP evolution from 4.3-Tahoe to 4.3-Reno," *Proc. of the* 18th *Internet Engineering Task Force*

[13] K. R. Krishnan, "The Hurst parameter of non-Markovian on-off traffic sources", Internal Bellcore Report, 1995.

[14] T. V. Lakshman and U. Madhow, "The Performance of TCP/IP for Networks with High-Bandwidth Delay Products and Random Loss", *IEEE/ACM Transactions on Networking*, pp. 336-350, June 1997.

[15] T. V. Lakshman, U. Madhow and B. Suter, "Window-based error recovery and flow control with a slow acknowledgement channel: a study of TCP/IP performance" *Proc. IEEE INFOCOM '97*, April 1997.

[16] T. V. Lakshman, A. Neidhardt and T. J. Ott, "The Drop from Front Strategy in TCP over ATM and its Interworking with other Control Features" *Proc. INFOCOM '96*, pp. 1242-1250

[17] Y. S. Lin and C. B. Shung, "Quasi-Pushout Cell Discarding", *IEEE Communications Letters*, vol. 1, no. 5, pp. 146-148, September 1997.

[18] S. McCanne, S. Floyd, K. Fall, "ns version 1 - LBNL Network Simulator", `http://www-nrg.ee.lbl.gov/ns/`

[19] V. Paxson, "End-to-End Routing Behavior in the Internet", *Proc. SIGCOMM '96*, pp. 25-38.

[20] B. Suter, T. V. Lakshman, D. Stiliadis, A. K. Choudhury "Efficient Active Queue Management for Internet Routers", *submitted to Interop 1998*.

[21] D. Stiliadis and A. Varma, "Design and analysis of Frame-based Fair Queueing: A New Traffic Scheduling Algorithm for Packet-Switched Networks", *Proc. of ACM SIGMETRICS '96*, pp. 104-115, May 1996.

[22] A. Varma and D. Stiliadis, "Hardware Implementation of Fair Queuing Algorithms for Asynchronous Transfer Mode Networks", *IEEE Communications Magazine*, pp. 54-68, December 1997.

[23] S. X. Wei, E. J. Coyle, and M. T. Hsiao, "An Optimal Buffer Management Policy for High-Performance Packet Switching," in *Proc. IEEE GLOBECOM '91*, (Phoenix, Arizona), pp. 924–928, Dec. 1991.

[24] L. Zhang, S. Shenker, and D. D. Clark, "Observations on the dynamics of a congestion control algorithm: the effects of two-way traffic," *Proc. ACM SIGCOMM '91*, pp. 133-147.