Scalable TCP Congestion Control

Robert Morris
Harvard University
rtm@eecs.harvard.edu

This paper appears in the Proceedings of the IEEE INFOCOM 2000 Conference.

Abstract—

The packet losses imposed by IP networks can cause long and erratic recovery delays, since senders must often use conservative loss detection and retransmission mechanisms. This paper proposes a model to explain and predict loss rates for TCP traffic. Based on that model, the paper describes a new router buffering algorithm, Flow-Proportional Queuing (FPQ), that handles heavy TCP loads without imposing high loss rates. FPQ controls TCP by varying the router's queue length in proportion to the number of active TCP connections. Simulation results show that FPQ produces the same average transfer delays as existing schemes, but makes the delays more predictable and fairer.

I. Introduction

The traditional role of buffer space in Internet routers is to absorb transient imbalances between offered load and capacity. Choosing the amount of buffer memory has been something of a black art: too little risks high loss rates and low link utilization, too much risks long queuing delays. Current practice favors limiting buffer space to no more than is required for good utilization. The result is that routers control congestion by intentionally discarding packets when their queues are too long. Most Internet traffic sources respond to loss by decreasing the rate at which they send data, so limited buffering and loss feedback may appear to be a reasonable approach to congestion control.

In contrast, this paper argues that router queue lengths should vary with the number of active connections. The TCP protocol that controls sources' send rates degrades rapidly if the network cannot store at least a few packets per active connection. Thus the amount of router buffer space required for good performance scales with the number of active connections. If, as in current practice, the buffer space does not scale in this way, the result is highly variable delay on a scale perceptible by users.

The simultaneous requirements of low queuing delay and of large buffer memories for large numbers of connections pose a problem. This paper suggests the following solution. First, routers should have physical memory in proportion to the maximum number of active connections they are likely to encounter. Second, routers should enforce a dropping policy aimed at keeping the actual queue size proportional to the actual number of active connections. This algorithm, referred to here as FPQ (Flow-Proportional Queuing), automatically chooses a good tradeoff between queuing delay and loss rate over a wide range of loads.

FPQ provides congestion feedback using queuing delay, which it makes proportional to the number of connections. TCP's window flow control causes it to send at a rate inversely proportional to the delay. Thus the combination of TCP and FPQ causes each TCP to send at a rate inversely proportional to the number of TCPs sharing a link, just as desired.

Simulations under heavy load show that FPQ produces a fairer distribution of delays than loss feedback: every transfer sees the same queuing delay in FPQ, whereas loss feedback sharply segregates transfers into unlucky ones (which see timeouts) and lucky ones (which do not). FPQ's delay feedback produces the same overall delay as the timeouts produced by loss feedback.

The rest of this paper has the following organization. Section II demonstrates that TCP is unfair and erratic under high loss rates. Section III presents a model explaining how network load affects loss rate. Section IV describes the FPQ algorithm for coping with heavy network loads. Section V evaluates FPQ's performance with simulations. Section VI relates FPQ to previous work in queuing and congestion control. Finally, Section VII summarizes the paper's results.

II. TCP Behavior with High Loss Rates

A TCP [1] sender sets the rate at which it sends data using window flow control. It ordinarily sends one window of packets per round trip time. TCP adjusts its window size to reflect network conditions as follows [2]. Each time TCP sends a window of packets it increases the window size by one packet. Each time TCP decides that the network has discarded a packet, it cuts the window in half. TCP can detect lost packets quickly using "fast retransmit" [3] as long as the window is larger than 3 packets. When fast retransmit fails TCP falls into a conservative retransmission timeout of a second or more. Thus loss affects delay in two ways: by decreasing TCP's window size and send rate, and by forcing TCP into timeouts.

Figure 1 shows the effect of loss on average transfer delay. The graph is the result of NS-1.4 [4] simulations at different uniform loss rates, using 20-packet transfers, TCP Tahoe [3], no delayed acks, and a 0.1 second round-trip propagation time. These results are similar to those produced by Cardwell's model for short connections [5].

There is nothing very surprising in Figure 1: as the loss rate increases, TCP slows down. Hidden by the averages, however, are significantly skewed distributions, illustrated by Figure 2. Each of the plots shows the cumulative distribution of the time required to complete simulated 20-packet transfers at particular loss rates. For example, with a loss rate of 10%, about 40% of transfers completed in less than a second; 5% of the transfers took more than 10 seconds. The 40% correspond to transfers that experienced no timeouts. The "steps" are caused by TCP's

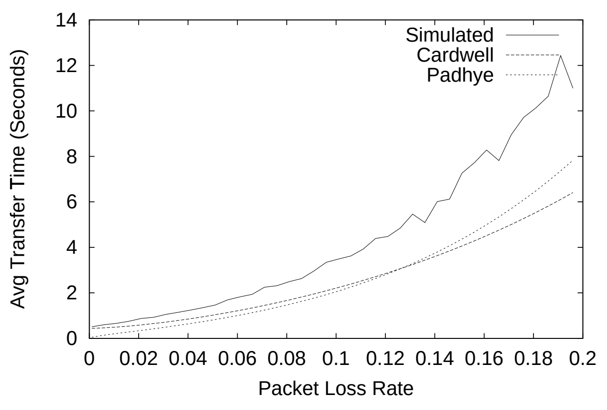

Fig. 1. Effect of loss on transfer delay. From simulations of 20-packet TCP transfers under uniform packet loss.

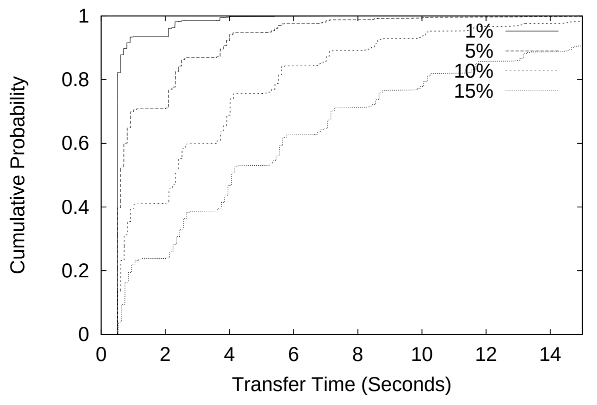

Fig. 2. Distribution of delays required for 20-packet TCP transfers. Each plot corresponds to a different uniform loss rate.

use of retransmission timers with 0.5-second granularity and by exponential retransmission backoff.

The delay distributions in Figure 2 are heavy tailed. For example, the distribution for a loss rate of 15% matches the Pareto distribution

$$P[X \leq x] = 1 - (4/x)^{1.8}$$

Figure 3 shows the closeness of the complements of the tails of the actual and Pareto distributions. The tail is heavy because TCP doubles its retransmission timeout interval with each successive loss. This means that when the network drops packets to force TCPs to slow down, the bulk of the slowing is done by an unlucky minority of the senders.

What we would like to see in Figure 2 is that, at any given loss rate, most transfers see about the same delay. This would provide fair and predictable service to users. In contrast, Figure 2 shows that high loss rates produce erratic and unfair delays. The first step towards improving the delay distribution is to understand the causes of high loss rates.

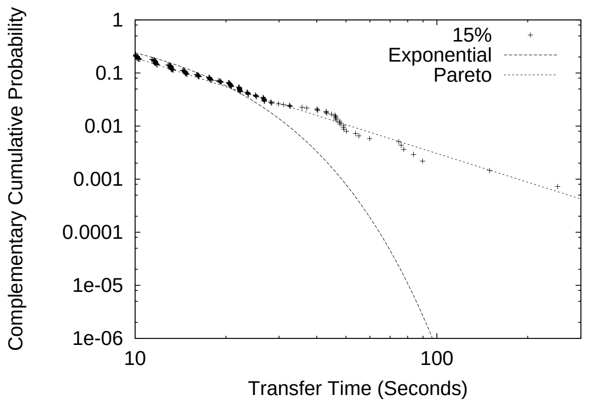

Fig. 3. Tail of the complementary transfer delay distribution for 15% loss rate, compared to a heavy-tailed Pareto distribution.

III. Load and Loss Rate

The following function approximates the average window size (w) that TCP uses when faced with a particular average loss rate (l):

$$w = \frac{0.87}{\sqrt{l}} \qquad (1)$$

This formula is adapted from Floyd [6] and Mathis *et al.* [7]; more detailed approximations can be found in Padhye *et al.* [8] and Cardwell *et al.* [5].

Equation 1 can be viewed in two ways. First, if the network discards packets at a rate independent of the sender's actions, Equation 1 describes how the sender will react. Second, if the network can store only a limited number of packets, Equation 1 indicates the loss rate the network must impose in order to make TCP's window fit in that storage. We can rearrange the equation to emphasize this view:

$$l = \frac{0.76}{w^2} \qquad (2)$$

As a simple example of the use of Equation 2, consider a single TCP sending through bottleneck router that limits the queue length to 8 packets. Ignoring packets stored in flight, the TCP's window must vary between 4 and 8 packets, averaging 6. Equation 2 implies that the network must discard 2.1% of the TCP's packets. The network uses this loss rate to tell TCP the correct window size.

Consider next N TCPs sharing a bottleneck link with queue limited to S packets. The TCPs' window sizes must sum to S, so that $w = S/N$. Substituting into Equation 2 yields an approximate loss rate prediction:

$$l = 0.76\frac{N^2}{S^2} \qquad (3)$$

For bottlenecks shared by many TCPs, Equation 3 suggests:

• The "meaning" that the loss rate conveys back to sending TCPs is the per-TCP share of the bottleneck storage. The losses a TCP encounters reflect the aggregate situation, not its own actions.

- Load, in the sense of "heavy load causes high loss rates," is sensibly measured by the number of competing TCP connections.
- A bottleneck's capacity, or ability to handle load, is determined by its packet storage.

A. Limitations

Equation 1, and all the equations derived from it, are only accurate with certain assumptions. The competing TCPs must be making transfers of unlimited duration. The sending applications must always have data available to send. The TCPs must all be limited only by the bottleneck under consideration.

The equations are not accurate if the loss rate is high enough that TCP's behavior is dominated by timeouts. Part of the point of this paper is that appropriate buffering techniques can help avoid such high loss rates.

The amount of storage available in the network actually includes packets stored in flight on the links as well as packets queued in routers. This number is the sum, over all the TCPs, of each TCP's share of the bottleneck bandwidth times the TCP's propagation round-trip time. If the TCPs have similar round trip times, the link storage is the bottleneck bandwidth times the round trip time. Thus the proper value for S in Equation 3 is this link storage plus the router's queue buffers.

A further restriction on S is that it must reflect the average number of packets actually stored in the network. In particular it must reflect the average actual queue length, which is not typically the same as the router's maximum queue length parameter. Thus Equation 3 must be tailored to suit particular queuing and dropping strategies, the subject of Section III-B.

Subject to the restrictions and corrections outlined above, Equation 3 produces predictions that closely match simulations [9].

B. Specialization to RED

Equation 3 must be modified to reflect any particular queuing system's queue length management policy. Consider, for example, Random Early Detection (RED) [10]. RED discards randomly selected incoming packets at a rate set by this simplified formula:

$$l = \frac{q_a}{max_{th}} max_p \tag{4}$$

q_a is the average queue length over recent time, max_{th} is a parameter reflecting the maximum desired queue length, and max_p is a parameter reflecting the desired discard rate when $q_a = max_{th}$.

If there are N competing TCPs, then each TCP's window size will average roughly $w = q_a/N$. We can calculate the discard rate required to achieve this w using Equation 2:

$$l = 0.76 \frac{N^2}{{q_a}^2} \tag{5}$$

Combining Equations 4 and 5 to eliminate l and simplifying results in this estimate of the average queue length:

$$q_a = \frac{0.91 N^{2/3} {max_{th}}^{1/3}}{{max_p}^{1/3}} \tag{6}$$

A RED router works best if q_a never exceeds max_{th}, so that the router is never forced to drop 100% of incoming packets. Equation 6 implies that the parameter values required to achieve this goal depend on the number of active connections. This is a formalization of an idea used by ARED [11] and SRED [12], which keep q_a low by varying max_p as a function of N. The observation that one could instead vary max_{th} and fix max_p is one way of looking at the FPQ algorithm presented in Section IV of this paper.

C. Discussion

Many existing routers operate with limited amounts of buffering. One purpose of buffering is to smooth fluctuations in the traffic, so that data stored during peaks can be used to keep the link busy during troughs. Recommendations for the amount of buffering required to maximize link utilization range from a few dozen packets [13] to a delay-bandwidth product [14].

Small fixed limits on queue length cause S in Equation 3 to be constant, and force the loss rate to increase with the number of competing TCP connections. To a certain extent this is reasonable: as the loss rate increases, each TCP's window shrinks, and each TCP sends more slowly. The rate (r) at which each TCP sends packets can be derived from Equation 1 by dividing by the round trip time (R):

$$r = \frac{0.87}{R\sqrt{l}} \tag{7}$$

The bottleneck router varies l in order to make the r values from the competing TCPs sum to the bottleneck bandwidth.

The problem with this approach is that TCP's window mechanism is not elastic with small windows. First, the granularity with which TCP can change its rate is coarse. A TCP sending two packets per round trip time can change its rate by 50%, but not by smaller fractions. Second, a TCP cannot send smoothly at rates less than one packet per round trip time; it can only send slower than that using timeouts. Worse, TCP's "fast-retransmit" mechanism [3] cannot recover from a packet loss without a timeout if the window size is less than four packets. If TCP's window is always to be 4 or more packets, its divide-by-two decrease policy means the window must vary between 4 and 8, averaging six packets.

TCP's inelasticity places a limit on the number of active connections a router can handle with a fixed amount of buffer space. One way to look at this problem is that the network must store at least six packets per connection. Another view is that TCP has a minimum send rate of six packets per round trip time, placing an upper limit on the number of connections that can share a link's bandwidth. The result of exceeding these limits is the unfair and erratic behavior illustrated in Section II.

IV. Flow-Proportional Queuing

TCP works well only if the network is willing to store roughly six packets per active connection. One way to satisfy this constraint is for all routers to be provided with enough buffer space to cope with the maximum number of expected connections. Connection counts, however, vary greatly from place to place and from time to time [15]. The penalty for providing more buffer space than required is excessive queuing delay, since TCP's window mechanism tries to keep all buffer space occupied. Thus without constant manual tuning this approach would require an uncomfortable choice between excessive timeout delays and excessive queuing delays.

What is needed, then, is a way to keep the queue length proportional to the actual number of active connections at any given time. A router could do this by imposing a constant packet loss rate, regardless of queue length. One would choose the loss rate so that Equation 1 resulted in a window size larger than the four packets required to avoid timeouts. Since each TCP would contribute a constant-sized window of packets to the router's queue, the resulting system should adaptively impose the minimum queue length required to keep the TCPs from falling into timeouts.

Imposing a constant loss rate would cause three problems. First, if the number of connections is small, the loss rate may need to be reduced to allow window sizes large enough to keep the link busy. Second, routers that are not bottlenecks should not impose any loss rate. Third, if typical paths through the network involve multiple bottlenecks, each such bottleneck should impose only its share of the total desired loss rate.

All three of these problems can be solved if each router maintains a count of the active connections sharing each of its links. This count can be used to detect the problems and correct the loss rate accordingly. The following sections describe the details as implemented in a queue control algorithm called "Flow-Proportional Queuing," or FPQ.

A. Counting Connections

FPQ counts active TCP connections as follows. It maintains a bit vector called v of fixed length. When a packet arrives, FPQ hashes the packet's connection identifiers (IP addresses and port numbers) and sets the corresponding bit in v. FPQ clears randomly chosen bits from v at a rate calculated to clear all of v every few seconds. The count of set bits in v approximates the number of connections active in the last few seconds. Figure 4 contains pseudo-code for this algorithm. The FPQ simulations in Section V use a 5000-bit v and a clearing interval (t_{clear}) of 4 seconds

The code in Figure 4 may under-estimate the number of connections due to hash collisions. This error will be small if v has significantly more bits than there are connections. The size of the error can also be predicted and corrected; see [9].

This method of counting connections has two good qualities. First, it requires no explicit cooperation from TCP. Second, it requires very little state: on the order of one bit per connection.

```
connection_count(packet $p$)
    $h \leftarrow H(p)$
    if $v(h) = 0$
        $v(h) \leftarrow 1$
        $N \leftarrow N + 1$
    $t \leftarrow$ current time
    $n_{clear} \leftarrow v_{max} \frac{t - t_{last}}{t_{clear}}$
    if $n_{clear} > 0$
        $t_{last} \leftarrow t$
        for $i \leftarrow 0$ to $n_{clear} - 1$
            $r \leftarrow random(0..v_{max} - 1)$
            if $v(r) = 1$
                $v(r) = 0$
                $N \leftarrow N - 1$
    return($N$)
```

Variables:

$v(i)$	Vector of v_{max} bits. $v(i)$ indicates if a packet from a connection with hash i has arrived in the last t_{clear} seconds.
N	Count of one bits in v.
t_{last}	Time at which bits in v were last cleared.
r	Randomly selected index of a bit to clear in v.

Constants:

v_{max}	Size of v in bits; should be larger than the number of expected connections.
t_{clear}	Interval in seconds over which to clear all of v.
$H(p)$	Hashes a packet's connection identifying fields to a value between 0 and v_{max}.

Fig. 4. Connection-counting pseudo-code.

```
target_queue($N$)
    $q_t \leftarrow \max\left(\frac{P}{2N-1}, 6N\right)$
    return($q_t$)

target_loss($q_t$, $N$)
    $l_t \leftarrow \min\left(\left(\frac{0.87}{\frac{q_t+P}{N}+1}\right)^2, 0.021\right)$
    return($l_t$)
```

Fig. 5. Pseudo-code to choose the target queue length and packet discard rate. The N parameter is the current count of active TCP connections, as computed in Figure 4. P is an estimate of the network's delay-bandwidth product in packets.

B. Choosing Target Queue Length and Drop Rate

Once an FPQ router knows how many connections are active, it chooses a target queue length in one of two ways. If the number of connections is relatively large, FPQ aims to queue six packets per connection. Six packets are just enough to avoid forcing connections into timeout. The corresponding target loss rate, derived from Equation 2, is 2.1%.

If there are too few connections, however, 6 packet buffers per connection may be too few to keep the link busy. For example, suppose that there is one connection, that the delay-bandwidth product is P packets, and that the router buffers up to S packets. The TCP's window size will vary between $(S+P)/2$ and $S+P$. If $S < P$, the TCP won't be able to keep the link busy after each window decrease. With N connections, there must be enough buffer space that $S \geq (S+P)/2N$.

Figure 5 summarizes the calculations required to choose a target queue length (q_t). The corresponding target packet discard

```
adjusted_loss_rate($l_t$, $q_t$, $q_a$)
    $l_a \leftarrow l_t \frac{q_a}{q_t}$
    return($l_a$)
```

Fig. 6. Pseudo-code to adjust the discard rate in case the actual queue length diverges from the target. l_a is the adjusted discard rate. The parameters: l_t is the target loss rate, q_t is the target queue length, and q_a is the actual average queue length.

```
fpq(packet $p$)
    $N \leftarrow$ connection_count($p$)
    $q_t \leftarrow$ target_queue($N$)
    $l_t \leftarrow$ target_loss($q_t$, $N$)
    $q_a \leftarrow$ average measured queue length.
    $l_a \leftarrow$ adjusted_loss_rate($l_t$, $q_t$, $q_a$)
    if(random() $< l_a$)
        Discard the packet.
    else
        Enqueue the packet.
```

Fig. 7. Pseudo-code for FPQ's main packet handling routine.

rate is the rate that Equation 1 predicts will cause each of the N TCPs to use a window size of $(q_t+P)/N$. P is a parameter set by the network administrator to the product of the link bandwidth and an estimate of the typical connection's round trip time; its value is only important when tuning low-load situations.

C. Achieving the Targets

If the network consisted of a single bottleneck, simply applying the target loss rate calculated in Figure 5 should cause the actual queue length to match the target, q_t. If there are multiple bottlenecks, and each applies the target loss rate, each TCP will see a loss rate higher than intended. The result will be that each TCP's window will be too small, and the TCPs will tend to fall into timeout.

An FPQ router can detect this situation, since it will result in a queue length shorter than the target. The degree to which the actual queue is shorter than the target will reflect the number of bottlenecks. To correct for this effect, FPQ decreases the discard rate it applies in proportion to the ratio of the actual queue length to the target queue length. Figure 6 shows the algorithm for this adjustment.

Figure 6 is the real reason why FPQ must count the number of active connections. Without the count, FPQ would not be able to calculate the target queue length (q_t), and thus would not be able to correct the discard rate to reflect multiple bottlenecks.

D. FPQ as a Whole

Figure 7 shows FPQ's main packet processing routine, which mostly calls the subroutines defined in the previous sections. An FPQ router runs one instance of the algorithm for each of its output links.

The overall effect of a network of FPQ routers should be to cause each competing TCP to use a window of 6 packets. This is the minimum window that avoids frequent timeouts; thus the network will exhibit the minimum queuing delay consonant with good TCP performance.

FPQ controls TCP by varying the round trip time rather than by varying the loss rate. In terms of Equation 7, FPQ holds l constant and varies R in proportion to the number of competing connections. This causes each TCP to send at a rate inversely proportional to the number of connections, just as desired.

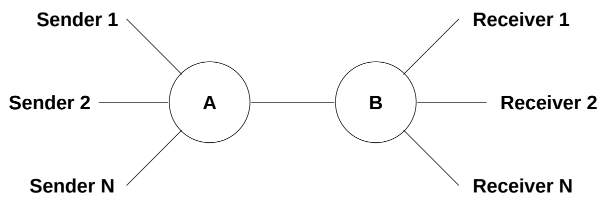

Fig. 8. Standard simulation configuration. Each of N TCP senders has its own host and its own link to router A. Router A connects to router B over a bottleneck link.

Packet size	576 bytes
Maximum window	64 kilobytes
TCP timer granularity	0.5 seconds
TCP delayed-ACK timer	0.2 seconds
TCP version	Tahoe
A to B propagation delay	45 milliseconds
A to B bandwidth	1 megabit/second
Sender i to A prop. delay	random, 0 to 10 ms
Sender i to A bandwidth	$10 \cdot (10/N)$ Mbits/sec
RED max_{th}	45 packets
RED min_{th}	5 packets
RED max_p	0.02
RED queue limit	no hard limit
Simulation length	500 seconds

Fig. 9. Summary of simulation parameters.

V. Evaluation

The main goal of FPQ is to improve the predictability and fairness of TCP transfers without increasing overall delay. This section uses simulations to demonstrate that FPQ achieves these goals.

The simulated network topology looks like Figure 8. The intent of this configuration is to capture what happens on heavily loaded links. For example, the link from A to B might be the link from an Internet Service Provider's backbone to a customer. Such links usually run slower than either the backbone or the customer's internal LAN, and thus act as bottlenecks.

The simulations use the NS 1.4 simulator [4]. Unless otherwise noted, they use the parameters given in Figure 9.

Each of the N senders makes repeated 10-packet transfers; after each transfer completes, the sender re-initializes its TCP state and starts another transfer. The simulations involve short transfers for two reasons. First, the average Internet TCP transfer is short [15]. Second, we can evaluate the fairness and predictability of network service by looking at the distribution of transfer latencies.

The configuration against which FPQ is compared is a RED [10] router. The router's max_{th} parameter is set to 45

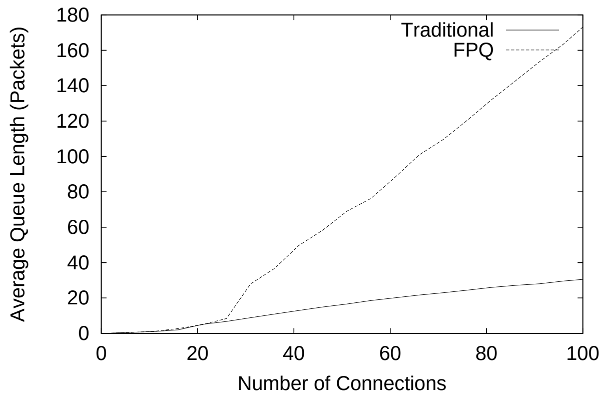

Fig. 10. Average queue length as a function of the number of competing TCP connections.

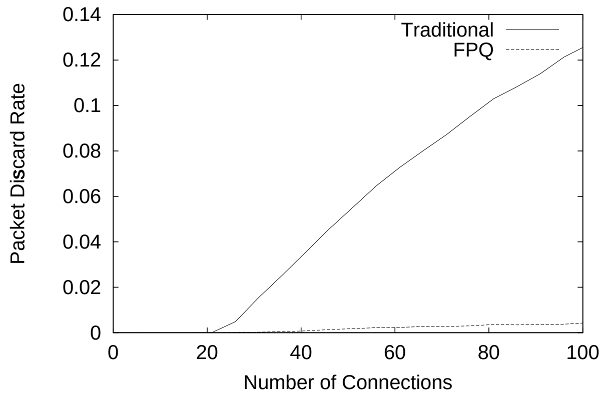

Fig. 11. Average drop rate as a function of the number of competing TCP connections.

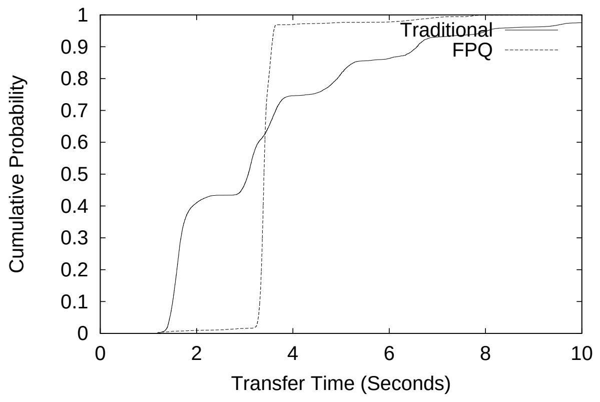

Fig. 12. Distribution of transfer times with 75 competing connections. Each connection makes repeated 10-packet transfers. Ideally all transfers would take 3.5 seconds.

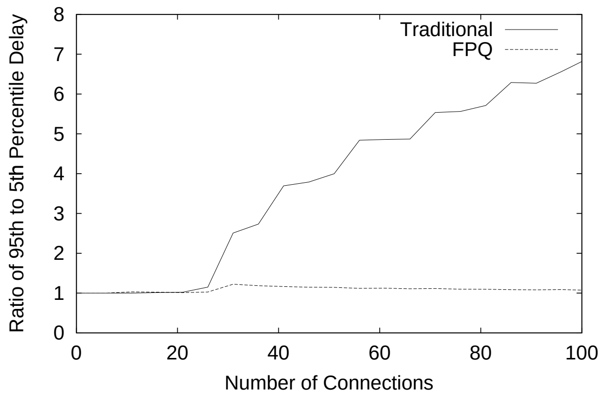

Fig. 13. Ratio of 95th to 5th percentile of delay as a function of the number of competing TCP connections.

packets, which is two delay-bandwidth products. The router's max_p is set to 0.02. This RED configuration will be referred to as "Traditional" to emphasize that it represents typical minimally buffered routers ([6], [14], [13], [16]) rather than (for example) RED with parameters tuned to use large numbers of buffers. RED performance can be improved for any particular situation by tuning its parameters, either manually or with automatic techniques such as ARED [11], SRED [12], or those described in this paper.

Figures 10 and 11 show the average queue length and discard rate, respectively, for the simulations. Each shows results for simulations with a range of numbers of competing connections. As expected, the traditional simulations exhibit increasing loss rate with increasing load, but maintain short queue lengths. FPQ, in contrast, lets the queue length grow with increasing load, but keeps the loss rate low. FPQ queues fewer than 6 packets per connection because the transfers are too short to allow large windows.

The average per-transfer delay produced by traditional and FPQ routers are the same, since average delay is a function only of transfer size, number of connections, and link bandwidth. The traditional and FPQ routers differ in the way they create the delay. The traditional router generates delay by forcing timeouts in response to dropped packets, whereas FPQ generates delay with queuing delay.

Figure 12 shows why it matters how the delay is generated. The figure compares the distribution of transfer times using traditional and FPQ routers. 75 connections share the 1-megabit bottleneck link, so the average time to transfer 10 576-byte packets should be 3.5 seconds. Both systems result in median transfer times of roughly 3.5 seconds. The traditional transfer times are quite skewed: 40% of transfers complete in less than half the fair time, and 10% complete in more than twice the fair time. With FPQ, however, almost all of the transfers see fair delays.

The unfairness caused by timeouts gets worse as the number of connections competing for a bottleneck increases, and thus FPQ's advantage increases as well. One way to capture overall unfairness is the ratio of the 95th to the 5th percentile of delays encountered by transfers in a particular simulation. Figure 13 plots this ratio as a function of the number of competing connections. The unfairness ratio increases for the traditional router because only a fixed number of connections can have packets in flight, and the remainder must wait in timeout. The ratio stays near one for FPQ because few connections time out.

To summarize, a traditional router experiencing heavy load will impose a low queuing delay but a high loss rate. The losses will force some unlucky transfers into timeout, allowing the lucky transfers to complete quickly. In contrast, a busy FPQ router will impose a high queuing delay but a low loss rate. Every transfer sees the same queuing delay, so the system is relatively fair and predictable.

VI. Related Work

The idea that window flow control in general has scaling limits because the window size cannot fall below one packet has long been known [17]. Villamizar [14] suggests that this might limit the number of TCP connections a router could support, and that increasing router memory or decreasing packet size could help. Eldridge [18] notes that window flow control cannot control the send rate well on very low-delay networks, and advocates use of rate control instead of window control; no specific mechanism is presented. TCP uses exponentially backed off retransmit timeouts as a form of rate control when window flow control fails; these timeouts help prevent congestion collapse but aren't fair.

Routers could notify senders of congestion explicitly rather than by dropping packets. Floyd [19] presents simulations of a TCP and RED network using explicit congestion notification (ECN). Floyd observes that ECN reduces timeouts for interactive connections, and thus provides lower-delay service.

Feng *et al.* [11] note that ECN works badly with small windows; a TCP with a window of 1 packet, for example, cannot conveniently slow down if it receives an ECN. As a solution, they implement Eldridge's rate-control proposal to allow TCP to send less than one packet per round trip time. The combination of rate-control and ECN increases the number of TCPs that can coexist with limited buffer space. Their rate increase algorithm is multiplicative: every time it sends a packet, it increases the rate by a fraction of itself. This increase policy turns out to have no bias towards fairness [20], so that ratios of send rates among connections will be stable. In contrast, TCP's standard window algorithms use an additive increase of one packet per round trip time, and do tend towards fairness.

When a RED router's queue overflows the router is forced to drop all incoming packets, and cannot drop a randomly chosen subset. Equation 6 shows that RED will persistently overflow if there are too many active connections. Two RED variants, ARED [11] and SRED [12], avoid overflows by effectively increasing RED's max_p parameter as the number of connections grows. ARED and SRED choose to keep the queue short and let the loss rate increase with the number of connections. FPQ's goal, in contrast, is to provide scalable buffering without increasing the loss rate.

Nagle [21] proposes that routers maintain a separate queue for each connection, with round-robin scheduling among the queues. The optimum sender behavior with this kind of fair queuing is to buffer just one packet in the router. If senders in fact buffered just one packet, routers could be built with unlimited buffer memory, but would only use it in proportion to the number of active connections. TCP senders do not have this optimal behavior, and per-connection queues are expensive; FPQ achieves much of the scaling benefit of Nagle's per-connection queuing using inexpensive FIFO queuing, though FPQ only makes sense for TCP traffic.

In an effort to achieve the benefits of Nagle's fair queuing without the overhead of maintaining per-connection queues, Lin and Morris [22] propose a fair dropping strategy called FRED. FRED stores packets in a single FIFO, but forbids any connection from queuing more than a handful of packets. This bounds the unfairness allowed, and requires only per-connection counting, not queuing. FRED should scale well with the number of connections because it allows a router to be built with a large packet memory, but only to use that memory in proportion to the current number of connections. However, FRED requires TCP to adopt more aggressive loss recovery algorithms [23] in order to avoid timeouts. FPQ avoids this tradeoff and also uses less per-connection state than FRED.

VII. Conclusions

The packet losses imposed by IP networks can cause long and erratic recovery delays, since senders often use conservative loss detection and retransmission mechanisms. This paper proposes a model to explain and predict loss rates for TCP traffic. Based on that model, the paper describes a router buffering algorithm that controls heavy TCP loads without imposing high loss rates.

The losses that a TCP sender experiences do not, in general, reflect that sender's send rate or choice of window size. Instead, the loss rate produced by a bottleneck router reflects the relationship between the total load on the router and the bottleneck's capacity. This paper quantifies that relationship by defining load as total number of competing senders and capacity as the number of packets that senders can store in the network.

If capacity is fixed, load and loss rate have a relationship somewhere between linear and quadratic. Most existing routers operate in this mode. As a result they control heavy loads by imposing high loss rates, long timeout delays, and erratic service.

This paper demonstrates a better approach to coping with load, called Flow-Proportional Queuing (FPQ). FPQ varies a bottleneck's capacity in response to the load in a way that causes the loss rate to remain constant. That is, FPQ monitors the number of active senders, and arranges that the average buffer space used in the router be proportional to that number. In effect FPQ controls TCP senders by varying queuing delay, rather than by varying the loss rate. FPQ imposes low queuing delay under low load. Under heavy load FPQ imposes the same average transfer time as loss-based control, but produces a fairer distribution of transfer times. Thus FPQ should improve the predictability and fairness of heavily loaded TCP networks.

Acknowledgments

I thank H. T. Kung for years of support and advice.

References

[1] W. Richard Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, Addison-Wesley, 1994.

[2] Van Jacobson, "Congestion avoidance and control," in *Proceedings of ACM SIGCOMM*, 1988.

[3] W. Richard Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," RFC2001, IETF, 1997, `ftp://ftp.ietf.org/rfc/rfc2001.txt`.

[4] Steve McCanne and Sally Floyd, "NS (Network Simulator)," `http://www-nrg.ee.lbl.gov/ns/`, June 1998.

[5] Neal Cardwell, Stefan Savage, and Tom Anderson, "Modeling the Performance of Short TCP Connections," Tech. Rep., University of Washington, 1998.

[6] Sally Floyd, "Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic," *Computer Communications Review*, vol. 21, no. 5, October 1991.

[7] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm," *ACM Computer Communication Review*, vol. 27, no. 3, July 1997.

[8] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," in *Proceedings of ACM SIGCOMM*, 1998.

[9] Robert Morris, *Scalable TCP Congestion Control*, Ph.D. thesis, Harvard University, Jan 1999.

[10] Sally Floyd and Van Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, August 1993.

[11] Wuchang Feng, Dilip Kandlur, Debanjan Saha, and Kang Shin, "Techniques for Eliminating Packet Loss in Congested TCP/IP Networks," Tech. Rep., University of Michigan, 1997, CSE-TR-349-97.

[12] Teunis Ott, T. V. Lakshman, and Larry Wong, "SRED: Stabilized RED," in *Proceedings of IEEE Infocom*, 1999.

[13] Sally Floyd, "RED: Discussions of Setting Parameters," `http://www-nrg.ee.lbl.gov/floyd/REDparameters.txt`, November 1997.

[14] Curtis Villamizar and Cheng Song, "High Performance TCP in ANSNET," *Computer Communications Review*, vol. 24, no. 5, October 1994.

[15] Kevin Thompson, Gregory Miller, and Rick Wilder, "Wide-Area Internet Traffic Patterns and Characteristics," *IEEE Network*, November/December 1997.

[16] Sean Doran, "RED on a busy ISP-facing line," April 1998, `ftp://ftp.isi.edu/end2end/end2end-interest-1998.mail`.

[17] Dimitri Bertsekas and Robert Gallager, *Data Networks*, Prentice Hall, Englewood Cliffs, New Jersey, 1987.

[18] Charles Eldridge, "Rate Controls in Standard Transport Layer Protocols," *Computer Communications Review*, vol. 22, no. 3, July 1992.

[19] Sally Floyd, "TCP and Explicit Congestion Notification," *ACM Computer Communication Review*, vol. 24, no. 5, October 1994.

[20] Dah-Ming Chiu and Raj Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Computer Networks and ISDN Systems*, vol. 17, pp. 1–14, 1989.

[21] John Nagle, "On Packet Switches With Infinite Storage," RFC970, IETF, 1985, `ftp://ftp.ietf.org/rfc/rfc0970.txt`.

[22] Dong Lin and Robert Morris, "Dynamics of Random Early Detection," in *Proceedings of ACM SIGCOMM*, 1997.

[23] Dong Lin and H. T. Kung, "TCP Fast Recovery Strategies: Analysis and Improvements," in *Proceedings of IEEE Infocom*, 1998.