Flow Control in High-Speed Networks with Long Delays

Srinivasan Keshav

AT&T Bell Laboratories,
600 Mountain Ave., Murray Hill, NJ 07974.
keshav@research.att.com

Abstract

The flow control component of a transport layer protocol regulates the natural data transmission rate of an application to match the service rate offered by the network. In this paper, we study the problems that arise when a flow control protocol has to deal with long delays in receiving information about network state, and has large amounts of data transmitted but unacknowledged.

We describe three representative flow control protocols, and study their behavior on a suite of three benchmark scenarios. Our simulations indicate that in networks with large delays, protocols that are insensitive to the state of the network, or that require multiple round trip times to attain the optimal transmission rate, will perform poorly. The packet-pair protocol, which avoids these problems, is shown to perform well under rather adverse conditions.

1. Introduction

We expect future high-speed networks to span entire continents, even parts of the globe. In such networks, the propagation delay introduced due to the limited speed of light significantly affects the performance of reactive flow control schemes implemented at the transport layer of the data sources. The fundamental problem is that any reactive flow control scheme receives information about network state that is old, and probably incorrect. As the delay increases, deciding on an appropriate flow rate on the basis of this incomplete information becomes harder, and wrong decisions can lead to congestion. Thus, it is important to study the behavior of flow control schemes in such situations.

The aim of this paper is to discuss the problems that arise with flow control in networks with a connectionless network layer in the presence of large delays. We present simulation results that compare the behavior of some representative flow control protocols in these networks. Since we assume that the network does not reserve bandwidth or buffers for each conversation, the onus of avoiding congestion is on the users, which places stress on the flow control protocol. We first describe the general problem. Then we compare three flow control protocols: (a) Generic, (b) BSD4.3-Tahoe operating system's TCP flow control scheme, and (c) PP, the packet-pair flow control scheme. They are evaluated in conjunction with two different packet scheduling disciplines: First Come First Served (FCFS) and Fair Queueing FQ [3]. We briefly describe each scheme, and then compare their behavior on some benchmark networks.

2. The flow control problem

The data stream from a user has application-specific dynamics. For example, file transfer like applications tend to send data at peak speeds that are far higher than a network can support. Other applications can send data in bursts that are larger than the available buffering at the endpoint or at intermediate queueing points. If a network allowed sources to send data at their natural rate, then the network (and destinations) could be subjected to wild fluctuations in sending rate, leading to excessive queueing delays, packet loss and congestion. In connectionless networks, the role of flow control is to modify the natural sending rate of an application to match the realities of network capacity, and to make the data stream better behaved. This is done by insisting that some assertions about the data stream are always valid, for example, that no more than 12 kbtyes of data will be outstanding (sent but unacknowledged) at any given time. The test of a flow control protocol is its effectiveness in making network operation smoother as a result of this modification.

The fundamental scaling dimension in the flow control problem is the bandwidth delay product [17, 18]. This is the amount of data that a source should keep outstanding in order to utilize network resources efficiently. When the bandwidth delay product (also referred to as the pipeline depth, or optimal window size; we use the symbol V) is small, then the flow control problem is not hard to solve. As long as queueing points have reasonable buffering capacity, and sources obey some kind of limit on the number of outstanding packets (window size), packet losses can be made rare, and the network operation is smooth. However, as V increases, the amount of buffers required at each switch becomes unreasonably large, and simple schemes, such as the one above, no longer work. Consider a simple example. Suppose source A is sending data on an otherwise idle network, where the end-to-end delay is 60ms, and the available end-to-end bandwidth is 6Mbps. Then, A will need to keep 45Kbytes outstanding to use the network efficiently. Now, suppose source B starts transmission, and takes up half of this bandwidth, i.e. 3Mbps. Since A knows about this change, and can react to it, only 60ms after the change, for 60ms, data from A is arrives at 6Mbps, and is served at 3Mbps, so that 22.5Kbytes of data build up in the queue. This leads to two problems: a) if A does not have 22.5 Kbytes worth of buffers at the switch, it will suffer packet loss. This is a rather large sized buffer to have, so packet loss is likely. b) If A does not realize that it has accumulated packets in its buffer, and immediately reduce its transmission rate, queue overflows will persist, leading to possible congestion. In short, A's flow control protocol must be *sensitive* to network state, and adjust its sending rate accordingly. Inflexibility on the part of the flow control algorithm can lead to severe performance problems.

In this discussion we have assumed that no bandwidth or buffers are reserved at a switch. If buffer reservations can be made, then the problem changes its character, and solutions such as those described in Reference [5] become applicable. Other intermediate buffer management schemes that combine sharing and reservation are described in [8]. However, since we restrict ourselves to connectionless networks, such schemes are beyond the scope of this paper.

Parts of this work appeared originally in Reference [13].

In high speed networks, a second source of problems is the presence of other users, who might be able to inject data at high speeds into some access link. The part of network that decides how traffic streams interact with each other is the scheduling discipline at the output queue of a switch. A discipline such as first-come-first-served (FCFS) is simple to implement, but it links the behavior of all the incoming traffic streams - if one source sends a lot of data, the single queue is filled, and all the sources have a higher probability of packet loss. Disciplines that implement per-conversation queueing either implicitly of explicitly (such as Round Robin or Fair Queueing) can get around this problem. As we shall see later, this delinking of sources helps in making flow control more effective.

To summarize, in a high speed network with large delays, problems can arise from two sources a) delay in knowing about network state can cause buffer buildups, and eventual congestion b) high speed sources can inject data rapidly into the network, causing problems for other sources. The second factor has been extensively studied in recent work [3, 16], and so we shall not consider it in any detail. Instead, we explore the first issue through simulations of some flow control protocols (in conjunction with scheduling disciplines). Similar studies have also been presented recently by [9, 17].

3. Description of protocols

A large number of flow control schemes have been proposed in the literature, but due to their number, it is not practical to study them all. Instead, it is illustrative to focus our attention on three flow control protocols that embody a range of techniques for flow control. The first protocol, that we call 'generic' represents the first attempt in designing flow control algorithms, and is relatively inflexible in its response to changes in the network state [19, 23]. The second protocol, the Jacobson-Karels modifications to TCP (JK) [6]. is an attempt to modify the basic framework of the 'generic' protocol to make it sensitive to network state.

The first two protocols do not depend on the choice of scheduling discipline at the intermediate queueing points in the network. In contrast, the third protocol, Packet-pair or PP, is designed for networks of round-robin like servers. It uses an analytic model for such networks to derive a control-theoretic basis for the flow control algorithm [12]. The idea is to design the protocol strictly from a formal model, so that its behavior is analytically tractable.

An ideal flow control protocol will match the current number of packets outstanding at a source to the current value of the delay pipeline depth (the product of the propagation delay and the rate of service at the bottleneck server). Call the current pipeline depth N. We describe the operation of these protocols and give an approximate analysis of their efficiency, in terms of the time taken to open the window to N starting from a window of size 1.

A generic version of source flow control [23] or in TCP (before 4.3 Tahoe BSD) [19], has two parts: sliding window flow control and the timeout mechanism. Sliding window flow control with a fixed size window is used to limit the number of packets outstanding from each source, so that the net buildup of packets at bottleneck queues is not excessive. This algorithm avoids queue build ups if the window sizes are small enough, and can fill the delay pipeline if the windows are larger than N, but, due to its inflexibility, cannot respond dynamically to changes in the network state. If this inflexibility leads to packet losses, then the timeout mechanism initiates recovery by retransmitting all unacknowledged packets. Timeout periods are set to βrtt where typically $\beta \sim 2$, and rtt is the exponentially averaged estimate of the round trip time.

With this algorithm, a source reaches a window size of N immediately, if this happens to be the value of the sliding window size. If not, the source will never reach N - the window will be either too large or too small.

The second flow control algorithm (JK) has the modification that the window size is allowed to change dynamically in response to changes in network state. The window size starts at 1, and is increased, first exponentially, and then linearly, till a timeout occurs, signalling a packet loss. At this point, the window is shutdown to either 1 (in BSD-4.3-Tahoe) or half the previous size (in BSD-4.3-Reno). Since the only information the source has about the network is a timeout, indicating overload, the window size oscillates around the correct operating point, which is bracketed by 1 and the window size at the point of timeout.

The algorithm takes approximately $\log_2(N) + N/2$ round trip times to open a window to size N. The algorithm works well when the time to open the window to the optimal size is small. But, when the delay increases, the time taken to reach the optimal window size can be large, and a source can lose throughput because of insufficiently sized windows.

The Jacobson/Karels flow control algorithm simulated here is defined by the 4.3BSD-Tahoe TCP implementation. This code deals with many issues unrelated to congestion control. Rather than using that code directly in our simulations, we choose to model the JK algorithm by adding many of the congestion control ideas found in that code, such as adjustable windows, better timeout calculations, and fast retransmit, to our generic flow control algorithm [6, 10].

Similar rate-based flow control protocols such as NETBLT [2] and the delay-based congestion avoidance scheme [7] allows users to increase and decrease their sending rates in response to changes monitored in the acknowledgment stream (instead of packet losses). The idea is that a slowed down acknowledgement rate implicitly signals congestion, and triggers a reduction in the source's sending rate. The delay-based congestion avoidance scheme reduces a source's window size whenever there is an increase in a congestion indicator which is computed using the round-trip-time delay. To a first approximation, an increase in the round-trip-time delay causes a reduction in the window size. We do not study these schemes in our simulations, since they are not widely implemented in current networks.

The control-theoretic Packet-Pair Protocol, PP, monitors the service rate at the slowest (bottleneck) server in the path from the source to the destination, and uses a simple control-theoretic algorithm to adapt to changes in the service rate [12, 14]. The idea is that the source sends out all data as pairs of back-to-back packets, and measures the spacing between the acknowledgments. It can be shown analytically that in networks of Fair Queueing (or Round-Robin like) servers, this spacing corresponds to the service rate at the bottleneck server [22]. An exponential average of the time series of such spacings is used to predict the current service rate, where the averaging constant is varied using a fuzzy controller [15]. Then, the current service

Label	Flow Control	Queueing Algorithm
G/FCFS	Generic	FCFS
G/FQ	Generic	FQ
JK/FCFS	JK	FCFS
JK/FQ	JK	FQ
PP/FQ	PP	FQ

Table 1: Algorithm Combinations

rate is chosen so that at the end of approximately one round trip time, given the predicted service rate, the queue size at the bottleneck will reach a setpoint [12]. The initial sending rate is decided by sending a pair of back to back packets and waiting one round trip time for their acknowledgments. Though rate control is used to choose the current sending rate, there is window limit as well, which prevents the flow control protocol from overflowing buffers even when are errors in rate monitoring. The time taken to reach a window size of N is approximately 2 round trip times, independent of the N.

The two scheduling disciplines that provide the environment for the study of these protocols are FCFS, and Fair Queueing (FQ). The FCFS discipline is standard in most current networks, and has been extensively analyzed. FQ is a simple extension to round-robin with per-conversation data queues that allows for variable sized packets [3].

4. Simulation results

This section presents a simulation results for a suite of three benchmark scenarios. The simulations were performed using the REAL simulator [11], using the methodology detailed in [13], and summarized below. In each scenario, we study a number of *protocol pairs*, where each pair is a choice of a flow control protocol and a switch scheduling algorithm from those described above. Though Packet-pair was designed for an environment where a source can reserve buffers and prevent packet losses, in this study, for the sake of comparison, such reservations are not assumed. The labels of the various test cases are given in Table 1.

The values chosen for the line speeds, delays and buffer sizes in the scenarios are not meant to be representative of a realistic network. Instead, they are chosen to accentuate the differences between the congestion control schemes. We choose to model all sources and switches as being infinitely fast. Thus, bottlenecks always occur at the output queues of switches.

In the scenarios, there are slow lines that act as bottlenecks, and fast lines that feed data to switches and bottleneck lines. Slow lines have a bandwidth of 80,000 bps, or 10 packets/sec. Fast lines have a bandwidth of 800,000 bps, or 100 packets/sec. All lines have zero propagation delay, unless otherwise marked. Sources are assumed to always have data to send, and are meant to model large file transfers (FTP protocol). The packet size is 1000 bytes. This roughly corresponds to the measured mean value 570 bytes in the DARPA Internet [1]. All the sources are assumed to start sending data at the same time - this makes throughput comparisions easy to make, and does not qualitatively alter our results.

Sinks acknowledge each packet received, and set the sequence number of the acknowledgment packet to that of the highest in-sequence data packet received so far. Acknowledgement packets traverse the data path in the reverse direction, and are treated as a separate conversation for the purpose of bandwidth allocation. The acknowledgement (ack) packets are 40 bytes long. The switches have finite buffers whose sizes, for convenience, are measured in packets rather than bytes.

Similar studies of the JK flow control scheme can be found in [9, 21, 24], and our results are qualitatively identical.

4.1. Scenario 1

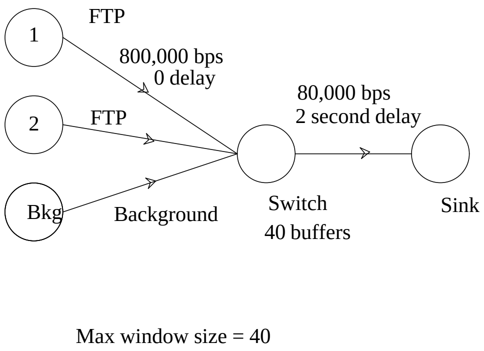

Figure 1: Scenario 1

Scenario 1 explores the effect of propagation delay in a simple topology. Two identical FTP sources send data through a bottleneck line that has a propagation delay of 2 seconds (though a delay of 2s seems rather large for a single link, since the link speed is slower than in a high speed network, the higher delay has the same overall effect on dynamics as a lower delay on a faster link). Cross traffic is modeled by a simple background source that sends data evenly spaced at a constant rate of half the bottleneck rate for 300 seconds, is idle for 300 seconds, and then resumes for 300 seconds. We expect the propagation delay to affect flow control protocols since changes in network state are detected only after some delay.

Simulation results are presented in Table 2 and Figures 1-3. The table shows the throughput rate excluding retransmissions, loss rate and retransmission rate (all in packets/sec) for each source when the background source is off, and when it is on. Since the simulation is almost completely deterministic, the values shown are for a single on or off period: the other periods are nearly identical. The figures show the dynamics of the flow control protocols in response to a change in the network state. To allow for easy comparison of the results with other studies, the time axis is marked in round trip times, and the window size axis in units of delay pipeline depth. So, a value of 0.5 on the window size axis corresponds to having a window size of half of the bottleneck rate multiplied by the round trip propagation delay, in this case, 0.5*10 packet/sec *4 sec = 20 packets.

The switch has 40 buffers. The bottleneck rate of 10 pkts/s, with a round trip propagation delay of 4 seconds gives an equivalent to 40 packets of storage on the link. Each source has a maximum window size of 40. Thus, when the background source is inactive, even if both sources open their window to the maximum, there is no packet loss (due to queue overflows) though spurious retransmissions, because of an overly small retransmission timer, are possible. When the background source is active, the number of buffers is no longer enough for all three sources. Since the background source is non-compliant (or ill-behaved), it can force the other sources to drop packets or cut down their sending rate. An ideal congestion control scheme

Background off						
	Throughput		Loss rate		Retransmission rate	
	1	2	1	2	1	2
G/FCFS	5.00	5.00	0	0	0	0
G/FQ	4.77	4.69	0	0	0	0.04
JK/FCFS	5.03	4.91	0	0	0	0
JK/FQ	4.94	4.92	0	0	0	0
PP/FQ	4.94	4.94	0	0	0	0

Background on									
	Throughput			Loss rate			Retransmission rate		
	1	2	Bkg	1	2	Bkg	1	2	Bkg
G/FCFS	4.94	4.94	0.12	0	0	4.87	0	0	0
G/FQ	1.85	0.39	4.49	.13	.19	.44	.13	.14	0
JK/FCFS	2.21	2.35	4.89	.09	.09	.83	.02	.02	0
JK/FQ	3.04	3.08	3.39	.10	0.14	1.52	0.03	0.03	0
PP/FQ	3.34	3.34	3.32	0	0	1.59	0	0	0

Table 2: Scenario 1 simulation results

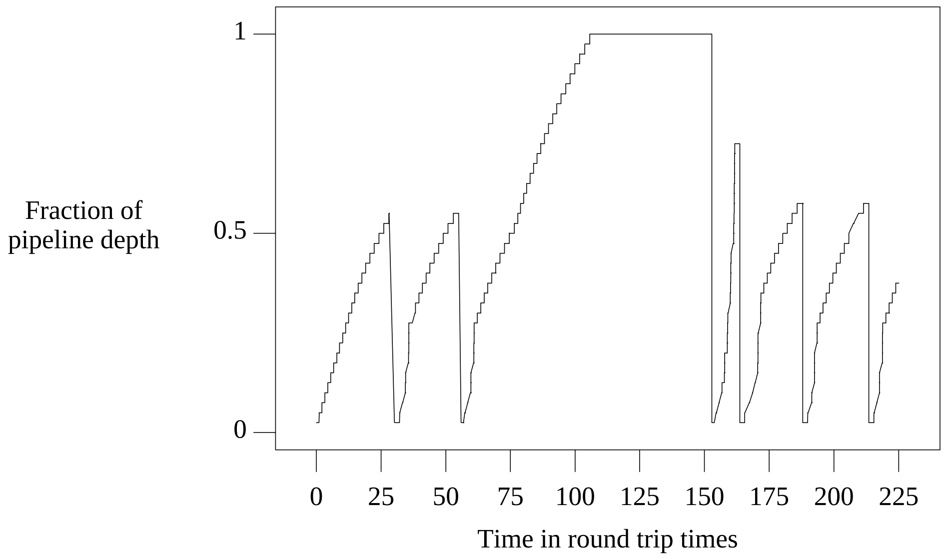

Figure 2: Scenario 1: JK/FCFS window dynamics for source 1

will allocate a throughput of 5.0 packets/s for each source when the background source is inactive, and a throughput of 3.33 packets/s otherwise.

With generic flow control and FCFS queueing, when the background source is inactive, there are no packet losses (this reflects on the careful choice of the window size - with some other window size, losses could occur). Since both the sources have the same window size, they share the bottleneck throughput exactly in half. Since the sources do not adjust their window size in response to changes in network state, the transition of the background source from *off* to *on* does not affect the window size, and full throughput is achieved (unlike other protocol pairs that take some time to increase their window in response to the state change, and hence lose throughput).

When the background source becomes active, it is after its inactive phase, and so it always finds the bottleneck buffer full. Hence, it drops almost all its packets, and the FTP sources split the bandwidth between themselves even when the background source is active.

When the scheduling discipline is FQ, matters are different. We discuss the situation when the background is active first. Here, the Generic FTP sources do not react to the presence of the background source, and hence keep their window at 40 packets. This causes packet losses and retransmissions. Thus, the background source is able to take up most of the bandwidth. This shows that, even with a fair bandwidth sharing scheduling algorithm, if the sources are insensitive to network state, the overall bandwidth allocation can be badly skewed. FQ cannot protect sources that adapt poorly to changes in network state.

Even when the background source is inactive, the FTPs still suffer from the effects from that source's previous active period. Hence, in this period, the FTPs share the throughput, though slightly unevenly. There are a few retransmissions that result from losses in the earlier period.

With JK flow control and FCFS scheduling, the situation is somewhat better. The window size vs. time diagram for source 1 (Figure 2) explains the behavior of the FTP sources. JK FTP sources open their flow control window, first exponentially,

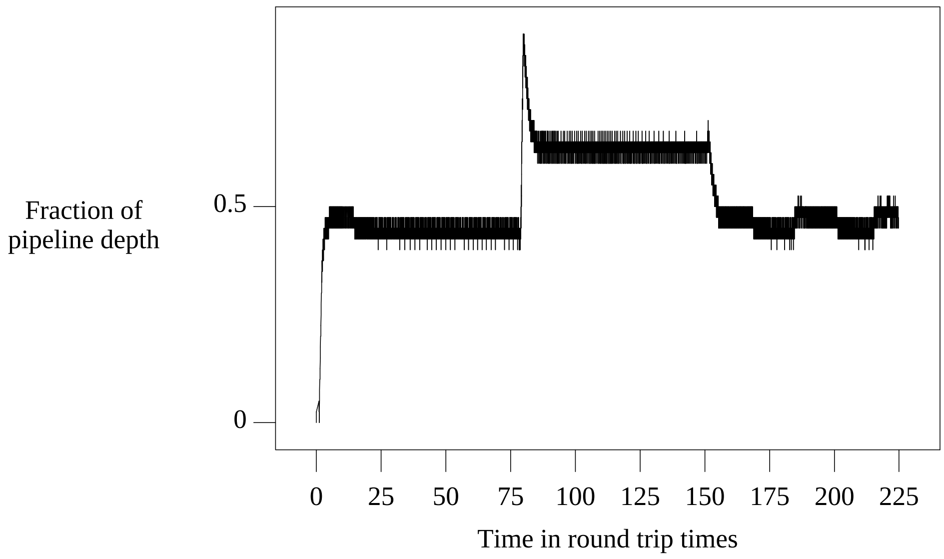

Figure 3: Scenario 1: PP/FQ Number of outstanding packets

and then linearly, until a packet loss causes the window size to drop to one. This cycle then repeats. We discuss the figure using the notation that 1 unit of window size refers to one bandwidth delay product (40 packets), and one unit of time refers to one round trip time (4 seconds).

When the background source is inactive, the window can open to its maximum of 1.0 without packet loss, and so between times 75 and 150 the window is stable at 1.0. In this region, the two FTP sources share bandwidth approximately equally. However, they may take a while to open their windows up in reaction to a change in the state, and so they could lose some throughput (for example, source 2 gets 4.91 packets per second instead of 5.0).

When the background source is active, it occupies some fraction of the buffers. This causes packet losses, and the FTP sources periodically shut down their window. Since the background source does not respond to packet loss, it gets much more throughput than the FTP sources (4.81 vs. ~2.2). Thus, non-conforming sources can adversely affect JK flow control if the scheduling algorithm does not provide protection.

When the scheduling algorithm is FQ, the dynamics are nearly identical, except that the FTP sources are protected from the background source. Thus, the background source is forced to drop packets due to its non-compliance, and the three sources share the bandwidth nearly equally. FTP sources have a few losses, but these are due to the intrinsic behavior of JK flow control. It is interesting to note, that contrary to intuition, even though the window size oscillates dramatically, the loss of throughput is not very significant. This is because a) the conversations are long lasting, so the loss of throughput at start up does not show up and b) the window is opened to a size that is much larger than optimal, and so the bottleneck server's buffer always contains some data, and even when the window is shut down, the bottleneck server almost always has data to send. However, short conversations that might send all their data by the time the window opens up fully would still be adversely affected by the multiple round trips it takes to fully open a window to the optimal size.

When PP flow control is used with FQ scheduling, matters are even better. The throughput when the background source is off is the same as with JK, and when it is on, is ~10% more. When the background source is inactive, the two FTP sources get almost half the bottleneck bandwidth each: the bandwidth loss is because it takes 2 round trip times for the sources to determine the correct window size. When the background source is active, the three sources share the bandwidth almost equally. The number of outstanding packets from source 1, which corresponds roughly to the window size, is plotted in Figure 3.

The figure reveals that when the background source is present, the 'window' is around 0.45 of the pipeline depth. This corresponds to about 18 packets outstanding. This makes sense, since the pipeline depth at this time is 40/3 = 13.3 packets, and the setpoint is 4 packets. Note that the window size oscillates rapidly with an small amplitude. This is the signature of the packet pair algorithm, since each time a pair is sent, this increases the number of packets outstanding by 2 packets. The other feature is a spike in the 'window' size at time 75, when the FTP source discovers the absence of the background source. The source immediately increases its sending rate to fill up the longer pipe. This spike, though large, occurs for such a small duration that it does not affect the overall sending pattern of the source, and so it is not a matter of much concern. A detailed examination of why the spike occurs, and how it affects flow control, is presented in the analysis accompanying Scenario 2.

4.2. Scenario 2

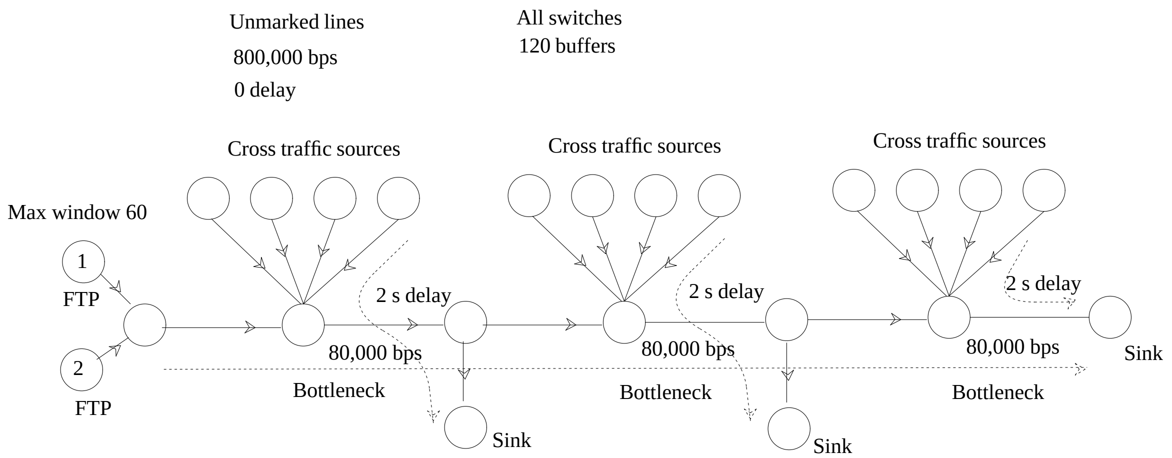

Figure 4: Scenario 2

Scenario 1 explored the behavior of the FQ algorithm and PP flow control in a network where there is little stochastic variance. Thus, the steady state is easily determined, and the flow control mechanism, once it determines a correct operating point, does not need to adjust to state changes. We would like to test the behavior of flow control mechanisms in the presence of rapid changes in network state. The performance of these mechanisms is explored in Scenarios 2 and 3.

Scenario 2 (Figure 4) explores problems that arise when there are large propagation delays, as well as three potential bottlenecks created by cross traffic from Poisson sources. We use multiple Poisson sources since with FQ, each source is mapped onto a separate service queue and this creates a larger variation in the service rate at the bottleneck. For FCFS, they could, in principle, be replaced by a single aggregate Poisson source. Due to the delays, sources receive outdated state information, and this can affect adversely the performance of a flow control algorithm. The three potential bottlenecks can lead to bottleneck migration, and large discrepancies in monitoring the bottleneck service rate.

It is generally accepted that a switch should have at least a bandwidth-delay product worth of buffers to be shared amongst the conversations sending data through that switch [4, 5]. Here, the minimum round trip propagation delay is 12 seconds, and the bottleneck bandwidth is 10 packets/s. Thus, 120 switch buffers are provided, as 120 is the bandwidth-delay product. Recall that in our simulations buffers are not reserved.

Each Poisson source has an average interpacket spacing of 0.5 seconds, so that, on average, it generates 2 packets/s, which is 20% of the bottleneck bandwidth. Since there are 4 Poisson sources, they can consume, on average, 80% of the bottleneck bandwidth. However, since there are 6 sources at each bottleneck, we expect FQ to restrict each Poisson source to roughly 16% of the bottleneck bandwidth, so they will have some packet losses.

The two PP sources are constrained by a maximum window size of 60 buffers. This is large enough to take up as much as half of the bandwidth, while we expect them to receive only one sixth, on the average. Since the sources are identically placed in the network, they should receive identical treatment. If this does not happen, then the congestion control scheme is unfair.

Scenario 2: Throughputs and delays				
	Throughput		Delay	
	1	2	1	2
G/FCFS	0.02 *0.52*	1.00 *0.59*	17.00 *22.94*	35.43 *7.51*
G/FQ	1.10 *0.48*	0.21 *0.13*	41.73 *11.05*	92.22 *56.65*
JK/FCFS	0.79 *0.06*	0.82 *0.12*	16.37 *1.06*	16.59 *1.04*
JK/FQ	1.62 *0.16*	1.72 *0.10*	13.53 *0.82*	16.41 *4.11*
PP/FQ	1.70 *0.02*	1.72 *0.01*	16.64 *4.97*	19.58 *4.37*

Scenario 2: Loss rate and retransmission rate				
	Loss rate		Retransmission rate	
	1	2	1	2
G/FCFS	0.05 *0.10*	0 *0.01*	0.31 *0.47*	0.12 *0.22*
G/FQ	0.02 *0.02*	0.05 *0.04*	0.03 *0.02*	0.06 *0.01*
JK/FCFS	0	0	0	0
JK/FQ	0	0.01 *0.03*	0	0
PP/FQ	0	0	0	0

Table 3: Scenario 2: simulation results

The simulation results are summarized in Table 3. Numerals in italics are standard deviations, that are computed by measuring the mean values over several large subintervals of the same simulation run.

The Poisson sources in this scenario are 'ill-behaved'; so, as expected, the Generic source pair does not do well in this scenario. Since the reasons for this have been examined earlier, we will only concentrate on the other three protocol pairs.

The JK/FCFS protocol pair does much better than G/FCFS, and this is because of its sensitivity to congestion. As the buffers in the bottlenecks build up, packet losses force window shutdown, preventing further retransmissions and losses. However, since the FTP sources are not protected from the Poisson sources, they lose packets because of misbehavior of the Poisson sources, causing window shutdown, and consequent loss of throughput. This is clear from the window vs. time diagram for source 1, Figure 5.

Note that the highest window size achieved is around 0.18, and, though the number of window shutdown events is small, the large propagation delay means that the time to open the window up again is large, and so each shutdown causes a possible loss of throughput (actual throughput loss will occur if the source does not recover by the time that the bottleneck queue dissipates).

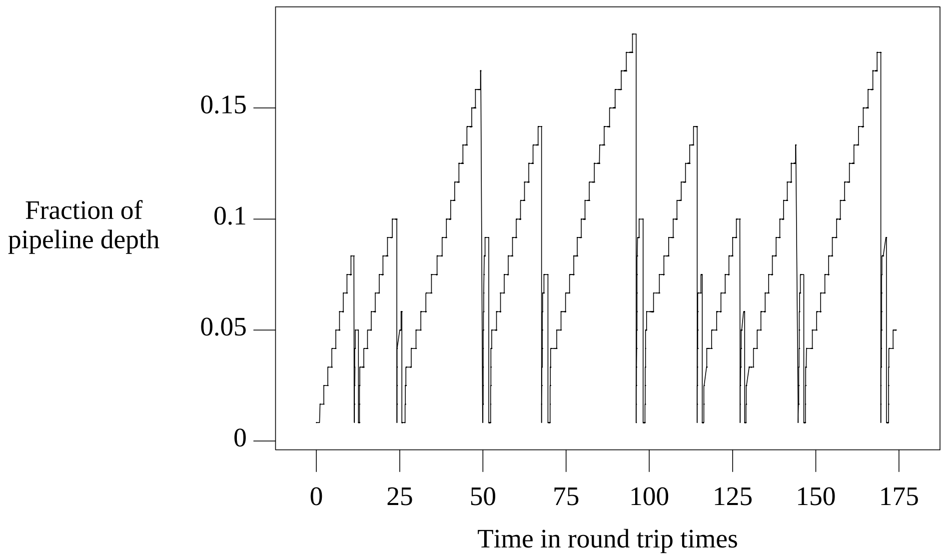

Figure 5: Scenario 2: JK/FCFS window vs. time

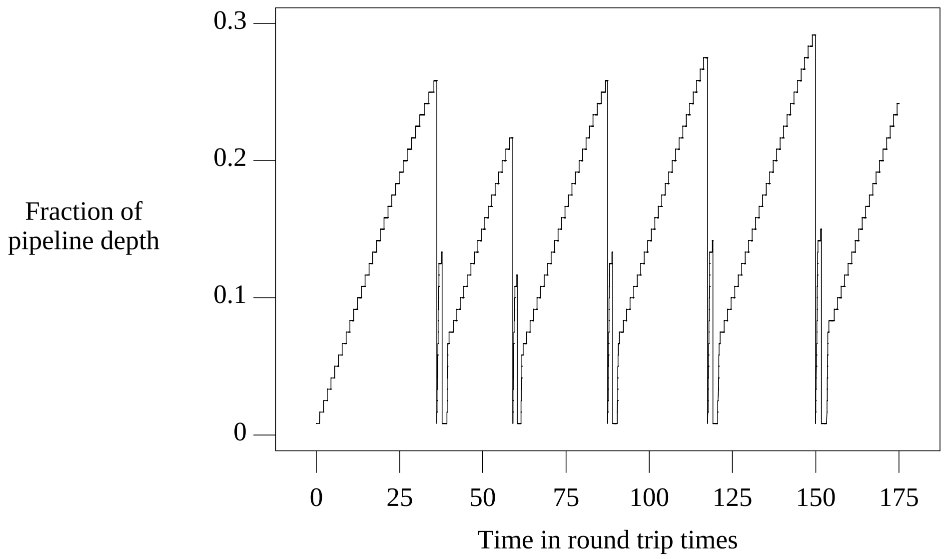

Figure 6: Scenario 2: JK/FQ window vs. time

When the scheduling discipline is changed to FQ, the situation improves considerably (Figure 6). The maximum window achieved is around 0.30, which indicates that the FTP sources have long periods of uninterrupted transmission. Both sources achieve almost their fair share of the throughput, which is 1.66 packets per second. There is some amount of unfairness, but this is due to the JK protocol, which is adversely affected by each shutdown.

The PP protocol can respond rapidly to changes in network state. Thus, if any of the Poisson sources is idle, the PP source can detect this, and make use of the idle time. Hence, the two PP sources obtain more than their fair share of the throughput (1.70 and 1.72 vs. 1.66) (Table 3). Moreover, the presence of multiple (and possibly migrating) bottlenecks, as well as the observation noise, does not affect the performance of the scheme. There are almost zero packet losses and retransmissions. We had earlier mentioned that PP does both rate-based and window-based flow control. The need for window limits is demonstrated by observing a trace of the number of packets outstanding vs. time fpr a small portion of the simulation period (Figure 7).

The figure shows that the number of outstanding packets shoots up rapidly, stops at 0.5, which is the window limit, and then decays slowly. This shape is explained below.

A rise in the number of outstanding packets is triggered when some Poisson sources are silent and the bottleneck has an idle period, so that a series of probes report a lower inter-ack value. When source 1 learns of this, it immediately increases its sending rate, and the number of outstanding packets rises steeply. The number of outstanding packets stabilizes at 0.5, which is the window limit. When a Poisson source becomes active again, the inter-ack spacing goes up, and further probes indicate that the bottleneck can no longer support the new sending rate. At this point the source cuts down its sending rate. But, for one RTT, while it is unaware of the lower service rate, it sends data much faster than the bottleneck can handle it, leading to a build

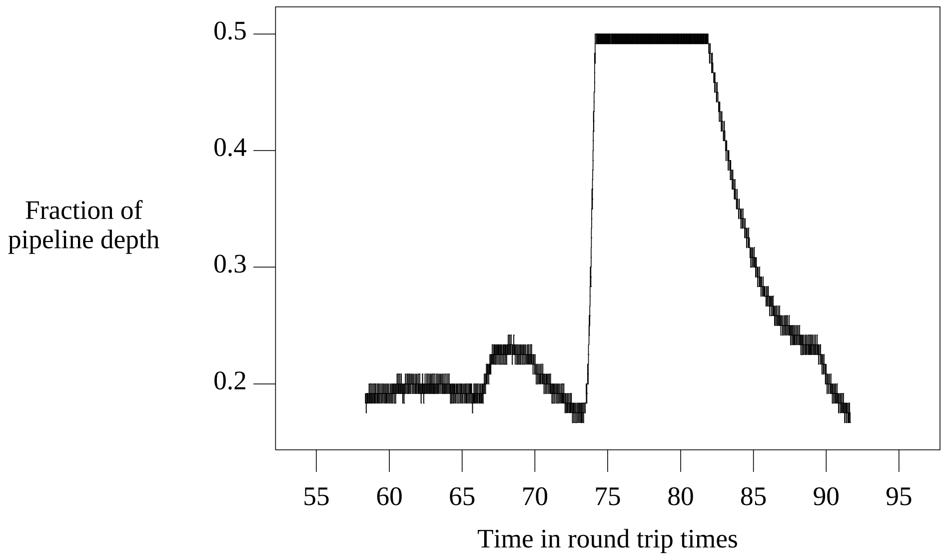

Figure 7: Scenario 2: Number of outstanding packets vs. time

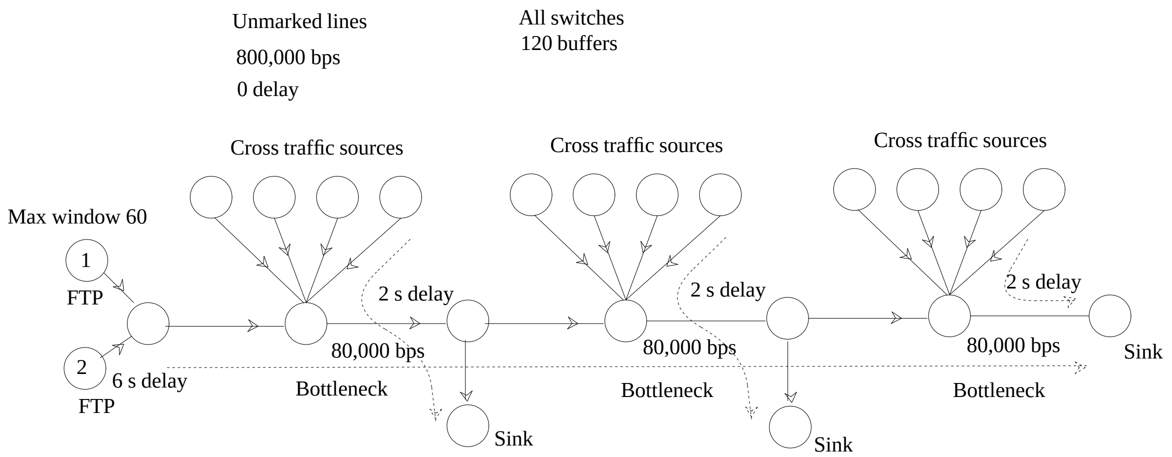

Figure 8: Scenario 3

up of a queue at the bottleneck. Note that the queues are built up quickly, since the source mistakenly sends data at a *higher* speed. However, the new bottleneck service rate is slower than this, so the queues drain slowly. In fact, even if the source sends no more packets, the number of outstanding packets will stay high. Thus, the slow decay of the curve.

This figure shows the usefulness of a window limit. In its absence, the source would send far too many packets in the RTT when it was misinformed, and would have had extensive packet losses. Here, even though we do not have buffer reservations, because of the window limit, there are no packet losses.

4.3. Scenario 3

Scenario 3 is similar to Scenario 2, except that source 1 has a round-trip-time delay of 12 seconds, and source 2, of 24 seconds (Figure 8). Thus, source 2 gets congestion information much later than source 1, and this can affect the fairness of the congestion control scheme. We examine the performances of the 7 protocol pairs in Table 4.

Scenario 3: Throughputs and delays				
	Throughput		Delay	
	1	2	1	2
G/FCFS	0.82 *0.67*	0.05 *0.75*	32.01 *9.78*	114.26 *108.91*
G/FQ	1.43 *0.49*	0.37 *0.48*	35.51 *7.05*	82.04 *91.36*
JK/FCFS	1.03 *0.20*	0.31 *0.10*	14.75 *0.46*	39.09 *0.64*
JK/FQ	1.39 *0.05*	0.86 *0.38*	13.52 *0.41*	36.88 *0.29*
PP/FQ	1.73 *0.02*	1.64	22.11 *3.48*	36.50

Scenario 3: Loss rate and retransmission rate				
	Loss rate		Retransmission rate	
	1	2	1	2
G/FCFS	0	0 *0.01*	0.03 *0.02*	0.43 *0.72*
G/FQ	0 *0.01*	0	0.04 *0.07*	0.36 *0.40*
JK/FCFS	0	0	0	0
JK/FQ	0	0	0	0
PP/FQ	0	0	0	0

Table 4: Scenario 3 simulation results

As in Scenario 2 the Generic protocol leads to poor performance, with many retransmissions (in fact, nearly 90% of the data transmission of source 2 is in the form of retransmissions!). The JK/FCFS and JK/FQ pairs both exhibit unfairness to the source with the longer RTT. This is because, on each packet loss, source 2 takes much longer to open its window than source 1. Thus, it loses throughput.

In contrast, the PP/FQ pair performs well, with no packet losses or retransmissions. The throughput allocation is almost fair, which is remarkable, considering that source 2 receives information that is rather stale. This scenario hence shows that PP behaves well even under fairly adverse conditions.

5. Conclusions

We have examined the performance of three representative flow control protocols in three benchmark networks with large bandwidth delay products. Our results show that protocols that ignore the state of the network, or have large start-up times, do not perform well in such networks. The PP flow control protocol consistently matches or outperforms the competing schemes, because of its short start up times, and ability to monitor network state.

Scenario 1 examined the dynamic behavior of flow control algorithms in response to an abrupt change in the network state. We saw that the JK protocol takes a while to respond to the change, while PP responds immediately. This is the reason why it does better. However, we note, that JK does better than one might expect, since the bottleneck's buffers are never allowed to completely empty.

Scenarios 2 and 3 test the robustness of the schemes under adverse conditions. In both scenarios, 'generic' performs poorly, while JK does well in Scenario 2, but not Scenario 3. PP does well in both scenarios, though the scenarios violate many of its design assumptions. In designing PP, it was assumed that the fluctuations in the probe value would be fairly small, whereas the changes in the probe value in Scenario 2 are as large as 10% and 30%. Second, there are three bottlenecks in tandem, so that bottleneck migration is possible, and can lead to errors in monitoring the bottleneck service rate. Third, there are no buffer reservations, as is recommended when using PP. Finally, there is a long propagation delay, so that the sources receive stale data. In spite of these difficulties, PP behaves rather well.

The adverse conditions of Scenario 2 are worsened in Scenario 3, where one source has double the propagation delay of the other. Whereas in Scenarios 1 and 2, JK/FQ did nearly as well as as PP/FQ, here, JK does not do well, since the conversation with longer delays takes a very long time to recover from each packet loss. We see that only PP is able to deliver reasonably fair throughput to the two sources in this scenario.

To summarize, we have shown that

- inflexible protocols such as 'generic' are unsuitable for high-speed networks with propagation delays.
- Schemes that involve a slow start phase, such as JK (and DECbit [20]) will discriminate against conversations with a long propagation delay, which will suffer loss of throughput.
- PP works well in the simulated scenarios, since it can rapidly adapt to changes in the network state.

These conclusions have been tested to the extent that our benchmarks are comprehensive. While we have tested for 'ill-behaved' users, Poisson cross traffic and multiple bottlenecks, we have ignored some other (perhaps equally important) factors such as: two-way traffic, bursty cross traffic, numerous short-duration conversations and the effect of conversations that start at random times. Thus, these limitations must be borne in mind while reviewing our conclusions. We recognize that no suite of benchmarks, at least at the current state of the art, can claim to be comprehensive. We have tried our best to create scenarios that test specific problems in congestion control schemes. It is possible that some of the factors we have ignored are critical in determining protocol performance, but this is still a matter for speculation. Developing a more comprehensive suite of benchmarks is a matter for future study.

To conclude, while our simulations are only for a small suite of scenarios, and each scenario only has a small number of nodes, we feel that results show several insights into designing flow control protocols suitable for high speed networks with large propagation delays.

6. Acknowledgments

I would like to thank A. DeSimone and S.P. Morgan for their thoughtful comments.

7. References

1. R. Caceres, P. B. Danzig, S. Jamin and D. J. Mitzel, Characteristics of Application Conversations in TCP/IP Wide-Area Internetworks, *Proc. ACM SigComm 1991*, September 1991.
2. D. D. Clark, M. L. Lambert and L. Zhang, NETBLT: A Bulk Data Transfer Protocol, RFC-998, Network Working Group, March 1987.
3. A. Demers, S. Keshav and S. Shenker, Analysis and Simulation of a Fair Queueing Algorithm, *Journal of Internetworking Research and Experience*, September 1990, 3-26;. also Proc. ACM SigComm, Sept. 1989, pp 1-12..
4. A. G. Fraser, Designing a Public Data Network, *IEEE Communications Magazine*, October 1991, 31-35.
5. E. L. Hahne, C. R. Kalmanek and S. P. Morgan, Fairness and Congestion Control on a Large ATM Data Network with Dynamically Adjustable Windows, *13th International Teletraffic Congress*, Copenhagen , June 1991.
6. V. Jacobson, Congestion Avoidance and Control, *Proc. ACM SigComm 1988*, August 1988, 314-329.
7. R. Jain, A Delay-based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks, *Computer Communications Review*, October 1989, 56-71.
8. F. Kamoun and L. Kleinrock, Analysis of Finite Storage in A Computer Network Node Environment Under General Traffic Conditions, *IEEE Trans. Comm. COM-28*, 7 (1980), 314-329.
9. H. Kanakia and P. P. Mishra, A Hop by Hop Rate-Based Congestion Control Scheme, *To Appear in Proc. ACM SigComm*, 1992.
10. P. Karn and C. Partridge, Improving Round-Trip Time Estimates in Reliable Transport Protocols, *ACM Trans. on Computer Systems 9*, 4 (November 1991), 364-373.

11. S. Keshav, REAL : A Network Simulator, Comp. Sci. Dept. Tech. Rpt. 88/472 , University of California, Berkeley, December 1988.

12. S. Keshav, A Control-Theoretic Approach to Flow Control, *Proc. ACM SigComm 1991*, September 1991.

13. S. Keshav, Congestion Control in Computer Networks, *PhD thesis* , University of California, Berkeley , August 1991.

14. S. Keshav, A. K. Agrawala and S. Singh, Design and Analysis of a Flow Control Algorithm for a Network of Rate Allocating Servers, in *Protocols for High Speed Networks II*, Elsevier Science Publishers/North-Holland, April 1991.

15. P. S. Khedkar and S. Keshav, Fuzzy Prediction of Timeseries, Proc. IEEE Conference on Fuzzy Systems-92, March 1992.

16. P. E. McKenney, Stochastic Fairness Queueing, *Proc. INFOCOM '90*, June 1990.

17. D. Mitra and J. B. Seery, Dynamic Adaptive Windows for High Speed Data Networks: Theory and Simulations , *Proc. ACM SigComm 1990*, September 1990, 30-40.

18. D. Mitra, Asymptotically Optimal Design of Congestion Control for High Speed Data Networks, *To Appear in IEEE Trans. on Communications*, 1991.

19. J. Postel, Transmission Control Protocol, RFC 793, USC Information Sciences Institute, 1981.

20. K. K. Ramakrishnan and R. Jain, A Binary Feedback Scheme for Congestion Avoidance in Computer Networks, *ACM ACM Trans. on Comp. Sys. 8*, 2 (May 1990), 158-181.

21. S. Shenker, L. Zhang and D. D. Clark, Some Observations on the Dynamics of a Congestion Control Scheme, *Computer Communications Review 20*, 5 (October 1990), 30-39.

22. S. Singh, A. K. Agrawala and S. Keshav, Deterministic Analysis of Flow and Congestion Control Policies in Virtual Circuits, Tech. Rpt.-2490, University of Maryland, June 1990.

23. Internet Transport Protocols, XSIS 028112, Xerox Corporation.

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 1991*, September 1991.