# Chapter 6: Simulation Experiments

## 6.1. Introduction

The previous chapters have presented the design and analysis of the Fair Queueing scheduling algorithm and the Packet-Pair flow control scheme. This chapter presents simulation experiments to study these mechanisms and to compare them with some others in the literature. The simulations also justify some of the claims and assumptions made earlier.

We first present the simulation methodology (§6.3) and then briefly describe some competing flow control proposals (§6.5). Finally, we present simulation results to compare and contrast the competing schemes (§6.6).

## 6.2. Why simulation?

Flow and congestion control schemes can be analyzed using several techniques, the most powerful of which are mathematical analysis and simulation. We used the former approach in earlier chapters. In Chapter 4 we used deterministic queueing analysis, and Chapter 5 presented a control-theoretic approach. Though mathematical modeling and analysis is a powerful technique, it has a few drawbacks. First, mathematical analyses make several simplifying assumptions that may not hold in practice. For example, the deterministic and stochastic models in Chapters 4 and 5 make strong assumptions about the bottleneck service rate, and the Fokker-Planck approximate analysis of reference [99] assumes that the bottleneck service rate is constant. Second, mathematical models ignore interactions that can prove to be critical in practice. For example, packet losses can trigger a spate of retransmissions, which, in turn, can cause further losses. In general, the sequence of events leading to congestion can be complex, and hard to analyze mathematically. For these two reasons, a simulation study of flow control protocols is not only useful, but often necessary.

In addition, the implementation of a protocol in a simulator brings out practical difficulties that are sometimes hidden in a formal approach, and this itself motivates new approaches. A good example of this is our use of fuzzy prediction in place of a mathematically adequate, but impractical, Kalman estimator.

## 6.3. Decisions in the design of simulation expermients

At least four decisions have to be made before undertaking a simulation study of flow and congestion control protocols. These are the choices of

- Level of detail in modeling
- Source workload
- Network topology
- Performance metrics

We discuss each in detail below.

### Level of detail

Some network simulation packages, such as IBM's RESQ, are strongly oriented towards queueing theoretic models. Sources are assumed to send data with interpacket spacings chosen from some standard distribution, such as the Erlang or exponential distribution. Users can only choose the form of the distribution, and cannot model the details of the congestion control algorithms. The effectiveness of a congestion control algorithm is often dependent on the implementation details. Since this is ignored by this approach, we feel that this level of modeling detail is inadequate.

Our approach, also used by others in the field [93, 117, 154], is to simulate each source and switch at the transport level. A source or switch is modeled by a function written in C, which mimics in detail the implementation of a congestion control protocol in, say, the UNIX kernel. The major difference between the model and the actual implementation is that, instead of

sending packets to a device driver for transmission, the model makes a simulator library call send() that simulates the transfer of a packet along a channel to another node. Similarly, instead of receiving packets from the data link layer, the model makes a blocking receive() call to the simulation package. Thus, the behavior of flow and congestion control algorithms is modeled in detail.

However, for simplicity, we ignore operating system issues, that, perhaps, are also significant. For example, we assume that, when a packet is forwarded by the network layer, it is put on the output trunk instantaneously. This ignores scheduling and bus contention delays. However, since these delays are on the order of tens of milliseconds, whereas the round trip time delays are on the order of hundreds of milliseconds, we feel that the error is not too large.

## Choice of source workload

A flow control mechanism operates on behalf of some user. How should user demands be modeled? Specifically, we would like to map each user to a series of packet lengths and a series of inter-packet time intervals. Two approaches are used in practice: a trace-driven approach, and a source-modeling approach [33].

In a trace-driven approach, an existing system is measured (traced), and the simulations use the traces to decide the values of variables such as the inter-packet spacing, and the packet size. However, the results from such a study are closely coupled to a few traces, and so they may not be generally valid. A stronger criticism is that traces measure a particular system, whereas they are used to simulate another system, with different parameters. For example, a trace may contain many retransmission events, due to packet losses at some buffer. If this is used to run a simulation where the buffer size is large enough to prevent losses, then the results are meaningless, since the retransmission events will not occur in the simulation. More insidiously, even packet generation times are closely linked to system parameters, and so the measured values cannot be used as the basis for the simulation of another closed system. A third objection is that the measured system is an existing one, whereas simulations are usually run for future systems that are still to be built. Hence the measured workload may not be a valid basis for simulating future systems. For example, current WAN workloads do not have any component due to video traffic, but this is expected to be a major component of future workloads.

Finally, traces can be extremely voluminous. If a large number of nodes are to be traced, the required storage can run into several gigabytes per day. Since simulations usually cannot use the entire trace, some part of the trace must be chosen as 'typical'. This choice is delicate, and hard to make.

Most of these objections can be overcome by measuring application characteristics, instead of transport layer timing details, as in reference [12]. In that approach, the volume of data is condensed by extracting histograms of metrics, and driving simulations by sampling these histograms with a random number. However, this tests the average case behavior of the congestion control scheme, while we are interested in worst case behavior. Thus, we do not use a trace-based workload model.

In the other approach, based on source modeling, an abstract source model is used to generate packet sizes and interpacket spacings. Typically, these values are generated from standard distributions such as the exponential and Gaussian distribution. We feel that this method is not realistic. Instead, our source models are loosely based on measurements of network traffic [13, 51, 105]. These studies indicate that:

1) the packet size distribution is strongly bimodal, with peaks at the minimum and the maximum packet size;
2) packet sizes are associated with specific protocols: mainly file-transfer-like protocols (FTP) and Telnet-like protocols (as explained in §1.6.7). We will call them 'FTP' and 'Telnet' for convenience, as before. 'FTP' sends data in maximum sized packets, and receives minimum sized acknowledgments. 'Telnet' sends data, and receives acknowledgments, in minimum sized packets.

Since FTP is used for bulk data transfer, we model FTP sources as sources that always have data to send, limited only by the flow control mechanism (infinite source assumption). The Telnet protocol sends one packet per user keystroke, and so this is modeled as a low data-rate source, with exponential interpacket spacing.

While our simulations mainly use FTP and Telnet workloads, there are a few other sources of interest. A Poisson source sends data with exponential interpacket spacing, and unlike a Telnet source, does not have any flow control. A malicious source sends data as fast as it can, and does not obey flow control. These sources are used to model cross traffic, as is explained in §6.6.4 and §6.6.7.

## Choice of network topology

We define network topology to be the number of sources and switches, their interconnection, and the sizes of various buffers in the switches. Network topologies can be chosen in at least three ways: to model an existing network, to model the 'average' case, or to model the worst case.

When simulations are meant to isolate problems in existing networks, it makes sense to simulate the existing topology. However, if the network is yet to be designed, then this approach is not useful.

Some researchers choose to simulate the average case. However, the notion of 'average' is poorly defined. Authors have their own pet choice, and it is not possible to reasonably compare results from different studies, or, for that matter, extrapolate the results from the chosen topology to another topology.

Our approach is to create a *suite* of topologies, each of which, though unrealistic by itself, stress tests a specific aspect of a congestion control mechanism. The contention is that one can better understand the behavior of these mechanisms by evaluating such behavior across an entire set of benchmarks, rather than on a single 'average' or existing topology (our choice of benchmarks is presented in section 6). While we do not claim to predict or test behavior in the average case, we can identify specific weaknesses in a congestion control mechanism, and this can be extrapolated to networks with similar topologies.

## Choice of metrics

The two main performance metrics used in flow control protocol evaluation are throughput and delay. Throughput refers to the number of packets sent from the source to the destination and correctly received, (of course excluding retransmissions), over some period of time. A packet's delay is the time taken by the packet to reach the destination from the source, or, more precisely, the time from the packet is handed to the network layer at the source to the time it is received at the transport layer at the destination. Delay sensitive sources (such as Telnet conversations) prefer low delays, and throughput sensitive sources (such as FTP conversations) prefer large throughputs. Thus, these measurements determine how far a congestion control mechanism can provide utility to the users.

In our simulations, we measure throughputs over *simulation intervals*. The time scale over which networks state changes is determined by visual inspection of state vs. time graphs, and an interval is chosen to be much larger than the time scale of network dynamics. Queueing delays are measured at each queueing point; round-trip delays are also measured. In addition, we measure the number of retransmissions and the number of packets dropped due to buffer overflows (in our simulations, there are no packet losses in transmission due to network errors such as bit corruption or line noise).

The metrics mentioned above are averages over one interval. Since simulation experiments typically are run for several simulation intervals, it is possible to calculate the standard deviation of the means over each interval. If each interval is 'long enough', then the distribution of the interval means of a metric can be approximated by a Gaussian distribution. Then, 95% confidence intervals for the mean of means can be calculated as thrice the standard deviation

on each side of the mean of means. This computation assumes that an interval is long enough that simulation dynamics are averaged out. While this can be achieved by linking the length of a simulation run to the confidence in the metrics [33, 103], we simply choose measurement intervals that are conservative enough that the standard deviation of the mean is fairly small, so that the confidence in the mean of means is high. Further, we ignore the mean over the first interval, so that the initial phase transient behavior is eliminated.

## 6.4. Simulator

All the simulation results presented here use the REAL network simulator. REAL is based on the NEST simulation package [4, 28] from Columbia University, and is described in detail in references [73, 74]. It provides users with the ability to specify network traffic workloads, congestion control algorithms, scheduling algorithms, and network topologies. These are then simulated at the application and transport levels at the sources, and the network layer at the switches, using event-driven simulation. The design decisions described in the previous section are reflected in REAL.

The user interface to REAL is through NetLanguage, a special-purpose language that describes the simulation scenario. Results are generated every simulation interval through reports that provide the means of the chosen metrics. At the end of a simulation run, the mean of means and the standard deviation of the metrics are reported. Users can also choose to plot the dynamics of any chosen variable, and these can be plotted on a display or a printer. The capabilities of REAL are illustrated in the simulations in this chapter.

## 6.5. Flow and congestion control protocols

We will present results from a simulation analysis of a few selected flow and congestion control protocols. A large number of congestion control schemes have been proposed in the literature, and these have been reviewed in Chapter 1. Due to their number, it is not practical to study them all. Instead, we focus our attention on three major flow control protocols that have been extensively studied in the literature, and also implemented in current networks. These are a generic transport flow control protocol [108, 150], the Jacobson-Karels modifications to TCP (JK) [63, 127, 154, 156], and the DECbit scheme [114-117]. We also study the control-theoretic version of the Packet-Pair protocol (PP_CTH) presented in Chapter 5. These protocols are briefly summarized below.

A generic version of source flow control, as implemented in XNS's SPP [150] or in TCP (before 4.3 Tahoe BSD) [108], has two parts. The timeout mechanism, which provides for congestion recovery, retransmits packets that have not been acknowledged before the timeout period has expired and sets a new timeout period. The timeout periods are given by $\beta rtt$ where typically $\beta \sim 2$, and $rtt$ is the exponentially averaged estimate of the round trip time (the $rtt$ estimate for retransmitted packets is the time from their first transmission to their acknowledgement). The congestion avoidance part of the algorithm is a sliding window flow control scheme, with a constant window size. The idea is that, if the number of packets outstanding from each source is limited, then the net buildup of packets at bottleneck queues will not be excessive. This algorithm is rather inflexible, in that it avoids congestion if the window sizes are small enough, and provides efficient service if the windows are large enough, but cannot respond adequately if either of these conditions is violated.

The second generation of flow control algorithms, exemplified by Jacobson and Karels' (JK) modified TCP [63] and the original DECbit proposal [15, 115-117], are descendants of the above generic algorithm, with the added feature that the window size is allowed to respond dynamically in response to network congestion (JK also includes, among other changes, fast retransmits in response to duplicate acknowledgements and substantial modifications to the timeout calculation [63, 71]). The algorithms use different signals for congestion; JK uses timeouts and duplicate acknowledgements whereas DECbit uses a header bit which is set by the switch on all packets whenever the average queue length is greater than one.

The second generation also includes rate-based flow control protocols such as NETBLT [17] and Jain's delay-based congestion avoidance scheme [65]. The NETBLT scheme, described in Chapter 4, allows users to increase and decrease their sending rates in response to changes monitored in the acknowledgment stream. 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.

In addition to the changes in the flow control protocol, some second generation flow control protocols are designed to work with selective congestion signaling. For instance, in the selective DECbit scheme and the Loss-load curve proposal [116, 148], the switch measures the flows of the various conversations and only sends congestion signals to those users who are using more than their fair share of bandwidth. The selective DECbit algorithm is designed to correct the previous unfairness for sources using different paths (see reference [116] and the simulation results below), and appears to offer reasonably fair and efficient congestion control in many networks.

Our simulations are for the selective DECbit algorithm based on the description in references [115, 116]. To enable DECbit flow control to operate with FQ switches, we developed a bit-setting FQ algorithm in which the congestion bits are set whenever the source's queue length is greater than ⅓ of its fair share of buffer space (note that this is a much simpler bit-setting algorithm than the DEC scheme, which involves complicated averages; however, the choice of ⅓ is completely *ad hoc*, and was chosen using performance tuning).

The Jacobson/Karels flow control algorithm is defined by the 4.3BSD TCP implementation. This code deals with many issues unrelated to congestion control. Rather than using that code directly in our simulations, we have chosen 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.

The control-theoretic Packet-Pair Protocol, PP_CTH, is implemented according to the details presented in Chapter 4, with the exception that the sending rate is computed using the fuzzy predictor and the once per probe rate computation as described in §5.6 and §5.7.

## 6.6. Simulation results

This section presents a suite of eight benchmark scenarios for which the congestion control schemes are evaluated. Scenarios 1 and 2 study the relative performance of FTP and Telnet sources. In the other scenarios, there are no Telnet sources, since their performance is qualitatively identical to what is obtained in the first two scenarios, and they have no appreciable impact on the performance of FTP sources.

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. The two scheduling algorithms studied are FCFS and FQ. The flow control protocols studied are Generic (G), JK, Selective DECbit (DEC) and PP_CTH. Though PP_CTH is 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 6.1.

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. (This assumption is not critical, since a network with slow switches can be converted to its dual with slow lines, with the only change being that output queueing in converted to input queueing. So, if we assume that the scheduling algorithms are run at the input queues instead of at the output queues, our simulation results hold unchanged.)

| Label | Flow Control | Queueing Algorithm |
|---|---|---|
| G/FCFS | Generic | FCFS |
| G/FQ | Generic | FQ |
| JK/FCFS | JK | FCFS |
| JK/FQ | JK | FQ |
| DEC/DEC | DECbit | Selective DECbit |
| DEC/FQbit | DECbit | FQ with bit setting |
| PP/FQ | PP_CTH | FQ |

*Table 6.1: Algorithm Combinations*

In the scenarios, there are slow lines that act as bottlenecks, and fast lines that feed data to switches and bottleneck lines. All lines have zero propagation delay, unless otherwise marked. The packet size is 1000 bytes for FTP packets, and 40 bytes for Telnet packets. This very roughly corresponds to measured mean values of 40 and 570 bytes in the Internet [12]. 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 the sources are assumed to start sending at the same time. The average inter-packet spacing for Telnet sources is 5 seconds. Both FTP's and Telnet's have their maximum window size set to 5 unless otherwise indicated (this is larger than the bandwidth delay product for most of the scenarios).

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. The small size of Telnet packets relative to FTP packets makes the effect of the FQ promptness parameter $\delta$ insignificant, so the FQ algorithm was implemented with $\delta$=0.

## Format of the results

Simulation results are presented as tables, and also in the form of throughput vs. delay plots (which we call *utility diagrams*). The tables present, for each scenario, and each protocol pair, the effective throughput (in packets per second), the mean round-trip-time delay (in seconds), the mean packet loss rate (in losses per second), and the mean retransmission rate (in packets per second). Throughputs and delays are written in the form X *Y*, where X is the mean of means over simulation intervals, and Y is the standard deviation around X. Variances are shown in oblique font, and values of less than 0.005 are omitted. Further, to improve readability, means of 0.0 are represented as 0. The numbers in the header are the numbers of the sources in the corresponding figure for each scenario.

An example of a utility diagram is presented in Figure 6.1. Here, we compare the simulation results for two sources, labeled 1 and 2, when the scheduling algorithm is FCFS or FQ. Each plotted number shows the throughput and round-trip-time delay experienced by the corresponding source (throughput values exclude retransmissions). The values are measured over a simulation run that lasts a number of simulation intervals (typically 7 intervals). Normal fonts are for results using FCFS scheduling, italics are for FQ. Some of the numbers are surrounded by boxes: the width of a box marks the 95% confidence interval in the measured throughput, and the height of a box marks the corresponding confidence in the delay. For the sake of clarity, if the confidence intervals are too small to be shown distinctly, the bounding rectangle is omitted.

In the example above, we note that with FCFS scheduling, source 1 receives a low queueing delay, and low throughput. The measured values of the mean throughput and delay for source 1 have high enough confidence that the bounding box is omitted. However, the

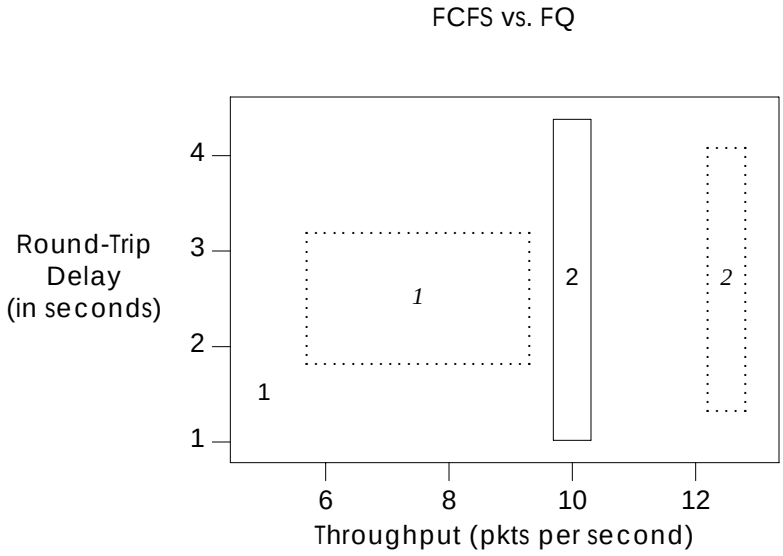


*Figure 6.1: Example of results*

confidence in the values for 2 is low, so the box is plotted. Note that with FQ, both sources receive more throughput, though source 1 also gets higher delay.

The utility diagram representation summarizes six dimensions of information - the means of delays and throughputs, their 95% confidence intervals, comparison between sources, and between scheduling disciplines. We feel that this novel representation of simulation results makes them easy to grasp.

## Results

### 6.6.1. Scenario 1

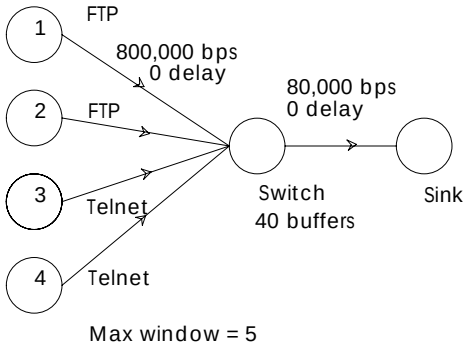


*Figure 6.2: Scenario 1*

This scenario is the simplest in the suite; it measures the performance of a congestion control scheme when it is under no stress. There are two FTP sources and two Telnet sources that send data through a single bottleneck switch. There are enough buffers so that, even when the FTP sources open their window to the maximum, there is no packet loss. The cross traffic is from the Telnet sources, which use little bandwidth and hence do not cause significant changes to the number of active conversations. Finally, the slow line does not have any propagation delay,

so the sources can adjust to state changes almost immediately.

We now examine the simulation results for the various protocol pairs for this scenario. These are summarized in Tables 6.2 and 6.3.

| Scenario 1 - FTP | | | | |
|---|---|---|---|---|
| Protocol Pair | Throughput | | Delay | |
| | 1 | 2 | 1 | 2 |
| G/FCFS | 4.99 | 4.99 | 1.00 | 1.00 |
| G/FQ | 4.99 | 4.99 | 1.00 | 1.00 |
| JK/FCFS | 4.99 | 4.99 | 1.00 | 1.00 |
| JK/FQ | 4.99 | 4.99 | 1.00 | 1.00 |
| DEC/DEC | 4.99 | 4.99 | 0.20 | 0.20 |
| DEC/FQB | 4.99 | 4.99 | 1.00 | 1.00 |
| PP/FQ | 4.99 | 4.99 | 1.48 | 1.22 |

*Table 6.2: Scenario 1 : Simulation results for FTP sources*

| Scenario 1 - Telnet | | | | |
|---|---|---|---|---|
| Protocol Pair | Throughput | | Delay | |
| | 3 | 4 | 3 | 4 |
| G/FCFS | 0.20 *0.02* | 0.21 *0.03* | 0.95 | 0.95 |
| G/FQ | 0.20 *0.02* | 0.21 *0.03* | 0.06 | 0.06 |
| JK/FCFS | 0.19 *0.03* | 0.18 *0.02* | 0.95 | 0.94 |
| JK/FQ | 0.20 *0.03* | 0.18 *0.04* | 0.06 | 0.06 |
| DEC/DEC | 0.20 *0.02* | 0.21 *0.03* | 0.14 | 0.14 |
| DEC/FQB | 0.20 *0.02* | 0.21 *0.03* | 0.06 | 0.06 |
| PP/FQ | 0.20 *0.02* | 0.21 *0.03* | 0.06 | 0.06 |

*Table 6.3: Scenario 1 : Simulation results for Telnet sources*

With generic flow control, both FCFS and FQ provide a fair bandwidth allocation (of the bottleneck capacity of 10 packets/second, Telnet sources get all that they can handle, and the FTPs get half each of the rest, nearly 5 packets/second). However, as the utility diagram in Figure 6.3 illustrates, FQ provides a much lower queueing delay for Telnets than FCFS does, without affecting the delay of the FTPs significantly. Since the interactive Telnet conversations gain utility from lower delays, this is a useful property of FQ.

Nearly identical results hold for the other protocol pair combinations, and so their utility diagrams are omitted. However, the selective DECbit flow control protocol gives a lower value for the Telnet delay than FCFS, since that flow control scheme is designed to keep the average queue length small. The results for PP_CTH illustrate two points. First, the setpoint is chosen to be 5 packets at the bottleneck, and so the RTT delay for PP_CTH FTPs is larger than for the other protocols, which keep their queue lengths smaller. However, since FTPs are not delay sensitive, this is of no consequence - in any case, lower delays can be obtained by choosing a lower setpoint (for example, with a setpoint of 2, the RTTs for sources 1 and 2 are .846 and .788 seconds respectively). Second, the protocol is sensitive to the initial estimate of the round trip propagation delay. Here, source 1 correctly estimates its delay, whereas 2 does not, since its first packet is queued behind source 1's first packet. Hence, it consistently overestimates the propagation delay, and consequently has a lower queue size, and a shorter RTT delay than source 1.

In all the simulations, there are no dropped packets, and no retransmissions, as expected.

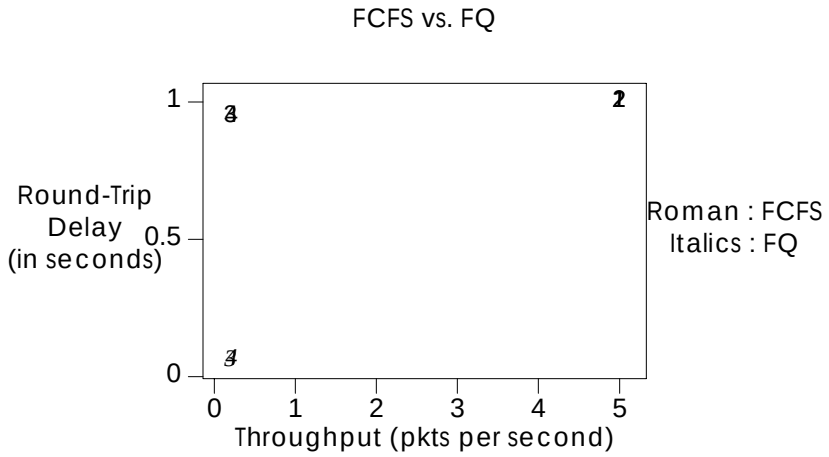


*Figure 6.3: Scenario1: Utility diagram for generic flow control (G)*

### 6.6.2. Scenario 2

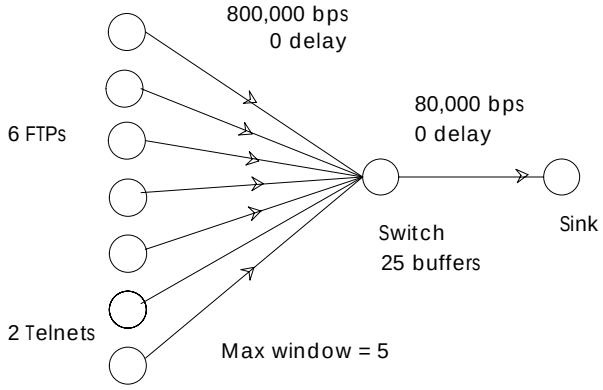


*Figure 6.4: Scenario 2*

This scenario has 6 FTP and 2 Telnet sources competing for a single line. The number of buffers at the switch (25) is such that if all the 6 FTPs open their window to the maximum (5 packets), there will be packet losses. Thus, they will experience congestion, and each source must cope not only with packet losses, but the reaction of the other sources to packet losses. This scenario also measures the resilience of the packet-pair probe. Since each PP_CTH source sends out data in the form of probes, it is possible that the probes may interact in non-obvious ways leading to a large observation noise.

The results of the simulation are presented in Table 6.4 and 6.5 and the corresponding utility diagrams in Figures 6.5-6.8. We discuss the results for each protocol pair below.

When FCFS switches are paired with generic flow control, the sources segregate into *winners*, which consume a large amount of bandwidth, and *losers*, which consume very little (Figure 6.5). This phenomenon develops because the queue at the bottleneck is almost always full. The ACK packets received by the winners serve as a signal that a buffer has just been freed, so their packets are rarely dropped. The losers retransmit at essentially random times, and thus have most of their packets dropped (the time when losers retransmit in relation to the winners, that is, their relative *phase*, plays a critical role in determining the exact form of the segregation.

| Protocol Pair | Throughput | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| G/FCFS | 1.69 *0.68* | 1.99 *0.06* | 1.09 *0.60* | 1.76 *0.33* | 1.37 *0.63* | 1.94 *0.11* |
| G/FQ | 1.46 *0.18* | 1.66 *0.10* | 1.57 *0.12* | 1.51 *0.15* | 1.64 *0.07* | 1.66 *0.14* |
| JK/FCFS | 1.80 *0.14* | 1.52 *0.50* | 1.90 *0.16* | 1.71 *0.44* | 1.49 *0.65* | 1.43 *0.49* |
| JK/FQ | 1.60 | 1.60 | 1.60 | 1.60 | 1.59 | 1.59 |
| DEC/DEC | 1.66 | 1.66 | 1.66 | 1.66 | 1.66 | 1.66 |
| DEC/FQB | 1.66 | 1.66 | 1.66 | 1.66 | 1.66 | 1.66 |
| PP/FQ | 1.66 | 1.66 | 1.66 | 1.66 | 1.66 | 1.66 |

| Protocol Pair | Delay | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| G/FCFS | 3.19 *1.59* | 2.50 *0.07* | 3.71 *1.45* | 2.57 *0.13* | 3.96 *2.83* | 2.52 *0.06* |
| G/FQ | 2.85 *0.29* | 2.56 *0.12* | 2.67 *0.19* | 2.75 *0.22* | 2.56 *0.09* | 2.55 *0.15* |
| JK/FCFS | 2.47 *0.04* | 2.47 *0.03* | 2.46 *0.04* | 2.44 *0.06* | 2.44 *0.04* | 2.45 *0.04* |
| JK/FQ | 2.37 *0.12* | 2.31 *0.11* | 2.31 *0.07* | 2.31 *0.16* | 2.13 *0.03* | 2.17 *0.02* |
| DEC/DEC | 0.60 | 0.60 | 0.60 | 0.60 | 0.60 | 0.60 |
| DEC/FQB | 1.09 | 1.11 | 1.12 | 1.14 | 1.11 | 1.11 |
| PP/FQ | 2.40 | 2.40 | 2.40 | 2.40 | 2.40 | 2.40 |

| Protocol Pair | Drop rate | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| G/FCFS | 0.02 *0.04* | 0 | 0.06 *0.03* | 0.02 *0.02* | 0.04 *0.03* | 0 *0.01* |
| G/FQ | 0.08 | 0.08 | 0.08 | 0.08 | 0.08 | 0.08 |
| JK/FCFS | 0.02 *0.01* | 0.04 *0.04* | 0.01 *0.01* | 0.02 *0.03* | 0.03 *0.04* | 0.04 *0.02* |
| JK/FQ | 0.06 | 0.07 | 0.07 | 0.07 *0.01* | 0.08 | 0.08 |
| DEC/DEC | 0 | 0 | 0 | 0 | 0 | 0 |
| DEC/FQB | 0 | 0 | 0 | 0 | 0 | 0 |
| PP/FQ | 0 | 0 | 0 | 0 | 0 | 0 |

| Protocol Pair | Retransmission rate | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| G/FCFS | 0.02 *0.04* | 0 | 0.06 *0.03* | 0.02 *0.03* | 0.04 *0.03* | 0 *0.01* |
| G/FQ | 0.08 | 0.08 | 0.08 | 0.08 | 0.08 | 0.08 |
| JK/FCFS | 0.02 *0.01* | 0.04 *0.04* | 0.01 *0.01* | 0.02 *0.03* | 0.03 *0.03* | 0.03 *0.02* |
| JK/FQ | 0.06 | 0.07 | 0.07 | 0.07 *0.01* | 0.08 | 0.08 |
| DEC/DEC | 0 | 0 | 0 | 0 | 0 | 0 |
| DEC/FQB | 0 | 0 | 0 | 0 | 0 | 0 |
| PP/FQ | 0 | 0 | 0 | 0 | 0 | 0 |

*Table 6.4: Scenario 2 : Simulation results for FTP sources*

This has been studied in depth by Floyd and Jacobson in [39,40]). Occasionally, a loser can place a packet in a buffer before a winner gets to it, and if this causes a winner to drop a packet, it becomes a loser. This alternation of winning and losing phases causes high variability in both the throughput and delay received by a source, and this is clearly seen in the utility diagram (Figure 6.5). In particular, source 3, which has a large loss rate and a high retransmission

| Scenario 2 - Telnet Throughputs and Delays | | | | |
|---|---|---|---|---|
| Protocol Pair | Throughputs | | Delays | |
| | 7 | 8 | 7 | 8 |
| G/FCFS | 0.05 *0.03* | 0.05 *0.04* | 2.30 *0.04* | 2.23 *0.10* |
| G/FQ | 0.20 *0.03* | 0.21 *0.03* | 0.06 | 0.06 |
| JK/FCFS | 0.08 *0.01* | 0.03 *0.06* | 2.30 *0.06* | 1.54 *1.09* |
| JK/FQ | 0.20 *0.03* | 0.21 *0.02* | 0.06 | 0.06 |
| DEC/DEC | 0.20 *0.03* | 0.21 *0.03* | 0.55 | 0.55 |
| DEC/FQB | 0.20 *0.03* | 0.21 *0.03* | 0.06 | 0.06 |
| PP/FQ | 0.20 *0.03* | 0.21 *0.03* | 0.06 | 0.06 |

| Scenario 2 - Telnet Drop rate and Retransmission rate | | | | |
|---|---|---|---|---|
| Protocol Pair | Drop rate | | Retransmission rate | |
| | 7 | 8 | 7 | 8 |
| G/FCFS | 0.06 *0.02* | 0.04 *0.02* | 0.04 *0.01* | 0.04 *0.01* |
| G/FQ | 0 | 0 | 0 | 0 |
| JK/FCFS | 0.05 | 0.03 *0.02* | 0.04 | 0.02 *0.02* |
| JK/FQ | 0 | 0 | 0 | 0 |
| DEC/DEC | 0 | 0 | 0 | 0 |
| DEC/FQB | 0 | 0 | 0 | 0 |
| PP/FQ | 0 | 0 | 0 | 0 |

*Table 6.5: Scenario 2 : Simulation results for Telnet sources*

rate, is a loser most of the time, while sources 2 and 6 are winners that enjoy more than their fair share of the bandwidth (1.66 pkts/sec). Note that sources 2 and 6 never have packet losses, whereas 1, 3, 4 and 5 have packet losses, which plunge them into losing phases. Further, Telnets, which also transmit at random time intervals, are shut out.

When generic flow control is combined with FQ, the strict segregation disappears, and the throughput available to each source, though variable, is more even overall. This is immediately obvious from the utility diagram. Also, note that Telnets get much lower delays, an effect noted in Scenario 1. The useful bandwidth (rate of nonduplicate packets) is around 90% of the net bottleneck bandwidth, while with FCFS it is nearly 100%. Both the variability in throughput and the underutilization of the bottleneck link are due to the inflexibility of the generic flow control, which is unable to reduce its load enough to prevent dropped packets. This not only necessitates retransmissions but also, because of the crudeness of the timeout congestion recovery mechanism, prevents FTP's from using their fair share of the bandwidth.

The combination of JK flow control with FCFS switches produces effects similar to those discussed for generic flow control (Figure 6.6). The segregation of sources into winners and losers is not as complete as with generic flow control approach, but all the sources exhibit winning and losing phases, and so experience high variability in throughput. Since the bottleneck queue is almost always full, the round trip delays show less variance. Note that the Telnets are shut out, as before. This is because the JK algorithm ensures that the switch's buffer is usually full, causing most of the Telnet packets to be dropped.

A detailed examination of the segregation phenomenon in similar scenarios has been carried out recently [39, 40]. This work shows that the appearance of segregation with JK FTP sources and FCFS switches depends strongly on the choice of the link delays, the maximum window size, and the buffer capacity at the switch. For some choices of these parameters (such as in our scenario) segregation happens, but there are many other choices for which the phenomenon is absent. Our aim is only to show that segregation is a possibility with the JK/FCFS

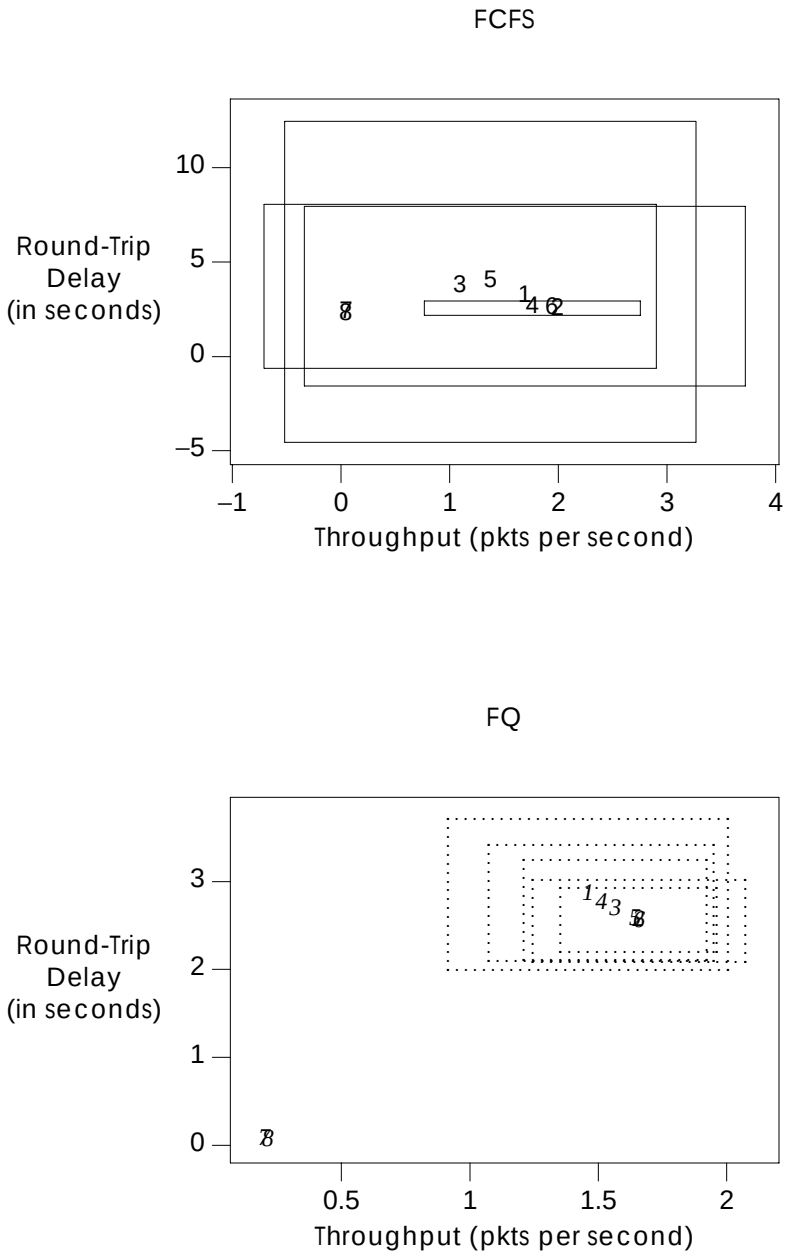


*Figure 6.5 Scenario 2: Generic flow control: FCFS vs. FQ*

protocol pair, and it should be borne in mind that this reflects our choice of parameter values.

In contrast, JK flow control combined with FQ produces reasonably fair and efficient allocation of the bandwidth, as shown by the corresponding utility diagram (Figure 6.6). The lesson is that fair queueing switches by themselves do not provide adequate congestion control; they must be combined with intelligent flow control algorithms at the sources. Also, note that, when FQ switches are used with either generic or JK flow control, the Telnet sources receive full throughput and relatively low delay.

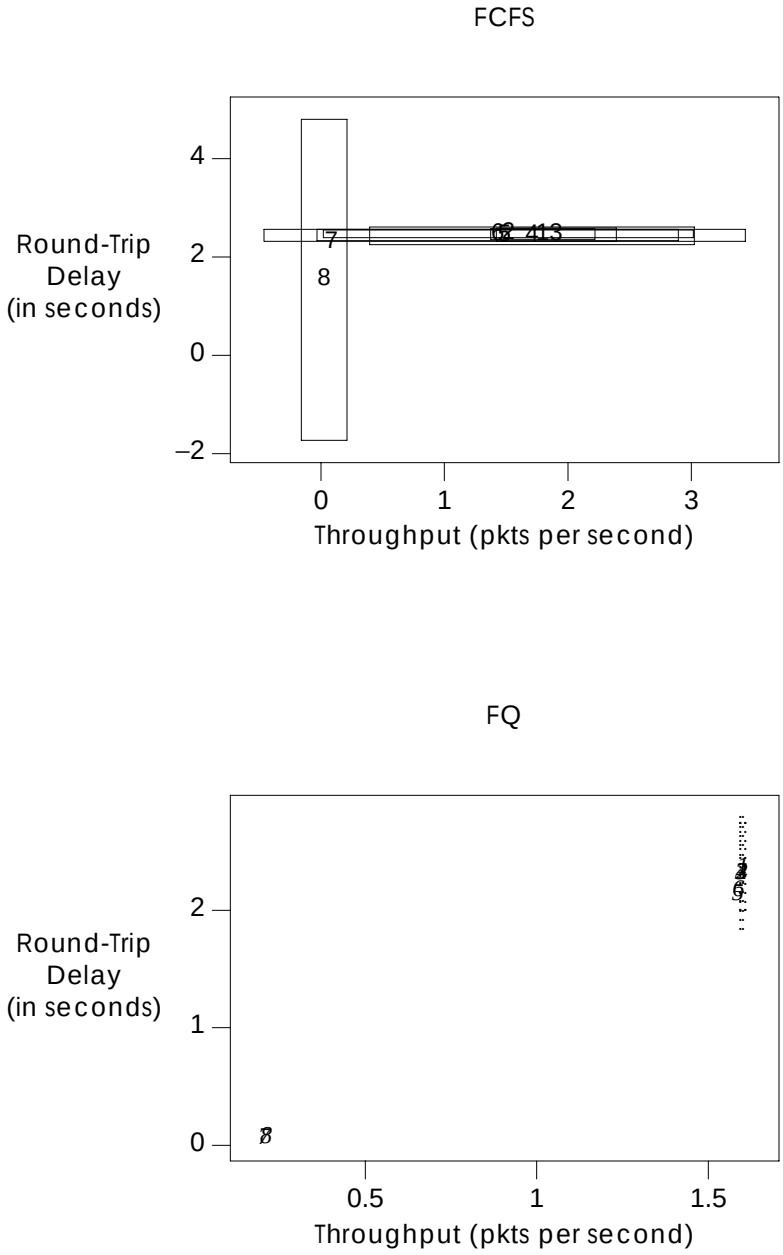


*Figure 6.6: Scenario 2: JK flow control: FCFS vs. FQ*

The selective DECbit algorithm manages to keep the bandwidth allocation perfectly fair, and there are no dropped packets or retransmissions. All the FTP sources receive identical throughputs and delays, and Telnets get a slightly lower delay (note that the Y axis on the utility diagram in Figure 6.7 is only from .54 to .60). The addition of FQ to the DECbit algorithm retains the fair bandwidth allocation and, in addition, lowers the Telnet delay by a factor of 9. The FTP delay does increase, but this should not decrease the utility of the FTP sources.

The PP_CTH FTP sources receive identical bandwidth, and this is exactly their fair share (Figure 6.8). There are no packet losses or retransmissions. The delays for the FTPs are higher than with JK, but they can be reduced by choosing a lower setpoint, as explained in Scenario 1. This

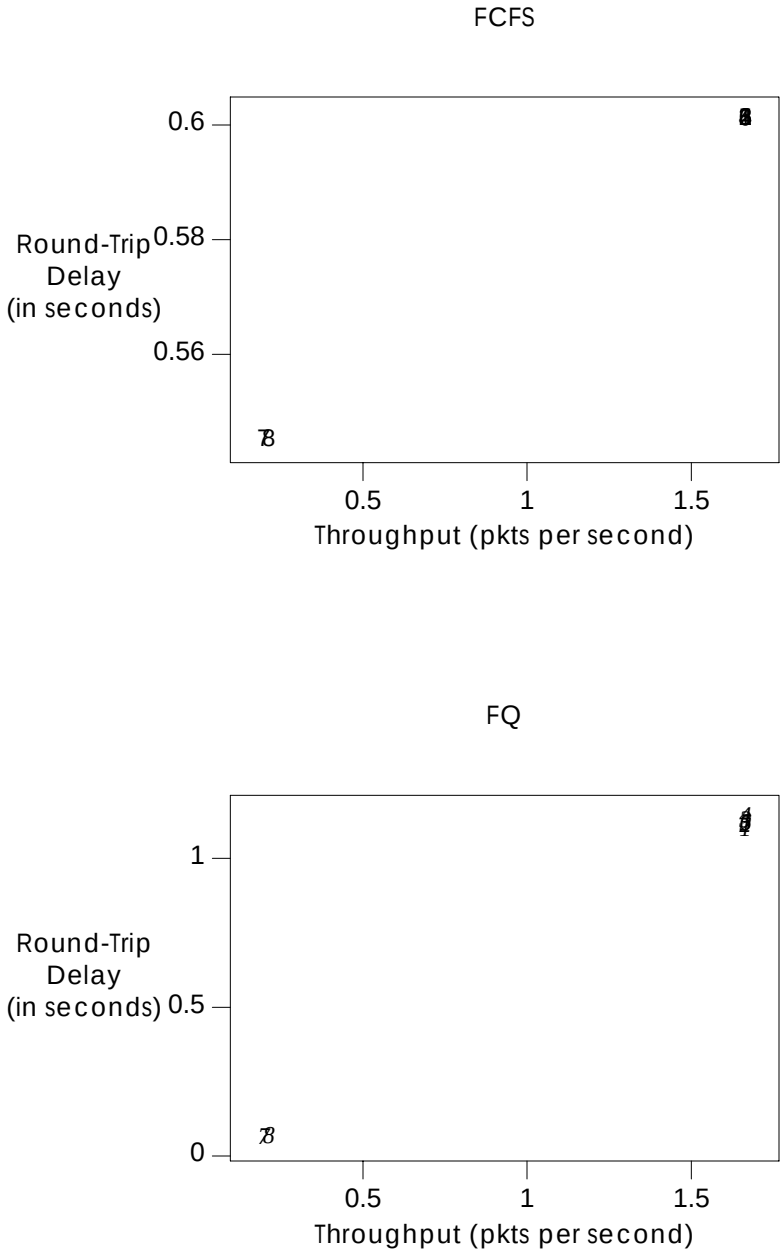


*Figure 6.7: Scenario 2: DEC flow control: Selective DECbit vs. FQbit*

simulation indicates that the probes used by PP_CTH do not, in fact, interfere with each other. Also, the effect of slightly different estimates in the propagation delay, due to the reliance on the first value of the probe, does not produce any appreciable difference in either the throughput or the delays received by the FTP sources.

Thus, we conclude that, for each of the first three flow control algorithms, replacing FCFS switches with FQ switches generally improves the FTP performance and dramatically improves the Telnet performance of this extremely overloaded network. We also noted some interesting phase effects that lead to segregation, staggered delay distributions and high variability in the throughput. Both FQ and PP_CTH show their effectiveness as congestion control schemes in

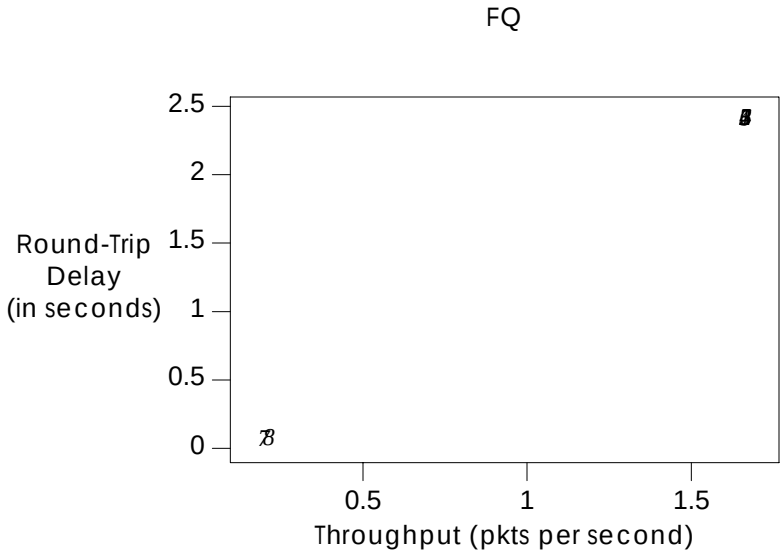


*Figure 6.8: Scenario 2: PP_CTH flow control*

overloaded networks.

### 6.6.3. Scenario 3

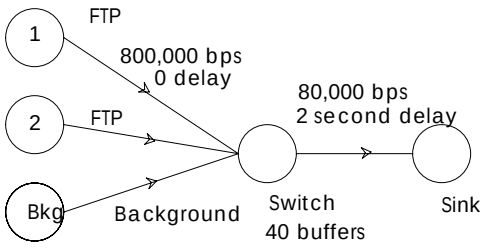


*Figure 6.9: Scenario 3*

Scenario 3 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. Cross traffic is modeled by a simple background source that sends data at 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 6.6 and Figures 6.10-6.13. The table shows throughputs, losses and retransmissions for each source for two situations: 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.

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

| Background off | | | | | | |
|---|---|---|---|---|---|---|
| | Throughput | | Drop 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 |
| J/FCFS | 5.03 | 4.91 | 0 | 0 | 0 | 0 |
| J/FQ | 4.94 | 4.92 | 0 | 0 | 0 | 0 |
| DEC/DEC | 3.84 | 3.48 | 0 | 0 | 0 | 0 |
| DEC/FQB | 4.90 | 4.90 | 0 | 0 | 0 | 0 |
| PP/FQ | 4.92 | 4.92 | 0 | 0 | 0 | 0 |

| Background on | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Throughput | | | Drop 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 |
| J/FCFS | 2.21 | 2.35 | 4.89 | .09 | .09 | .83 | .02 | .02 | 0 |
| J/FQ | 3.04 | 3.08 | 3.39 | .10 | 0.14 | 1.52 | 0.03 | 0.03 | 0 |
| DEC/DEC | 1.37 | 1.59 | 5.00 | 0 | 0 | 0 | 0 | 0 | 0 |
| DEC/FQB | 3.33 | 3.33 | 3.34 | 0 | 0 | 1.54 | 0 | 0 | 0 |
| PP/FQ | 3.35 | 3.35 | 3.23 | 0 | .05 | 1.65 | 0 | 0.06 | 0 |

*Table 6.6: Scenario 3 simulation results*

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 (though spurious retransmissions 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 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. 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 (this is similar to the segregation phenomenon described in in scenario 2, and is also sensitive to parameter choice). 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 (Figure 6.10) explains the behavior of the FTP sources.

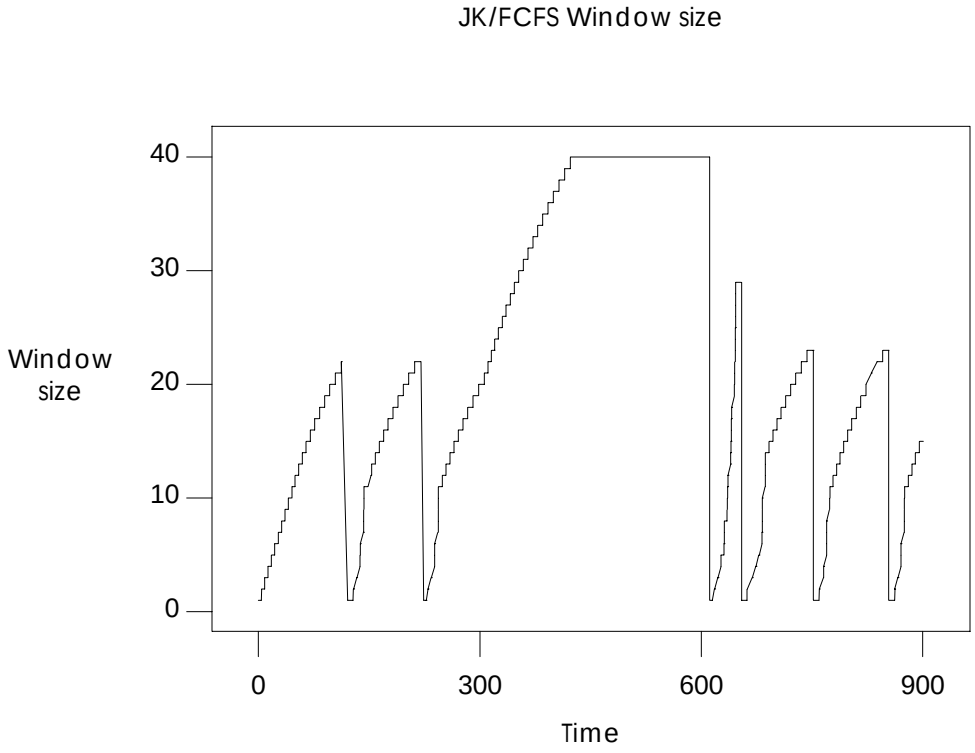


*Figure 6.10: Scenario 3: JK/FCFS Window dynamics*

JK FTP sources open their flow control window, first exponentially, and then linearly, until a packet loss causes the window size to drop to one. This cycle then repeats.

When the background source is inactive, the window can open to its maximum of 40 without packet loss, and so between times 300 and 600 the window is stable at 40. In this region, the two FTP sources share bandwidth approximately equally. However, they take a while to open their windows up in reaction to a change in the state, and so they lose some throughput.

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. 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 equally. FTP sources have a few losses, but these are due to the intrinsic behavior of JK flow control.

With selective DECbit congestion control, the non-compliance of the background source is a major problem. Even when that source is inactive, the long round-trip time delay implies that the sources take a long time to achieve the correct window size, thus losing throughput. By the

time they do reach the right size, the background source fires up, which drives the switch towards congestion. In response, the switch goes into panic mode, and sets bits on the FTP sources, which shut down their window to accommodate the background source. Thus, when the background source is active, it gets all the throughput it wants, and the FTP sources adjust themselves to its presence. This is clear from the window vs. time diagram (Figure 6.11).

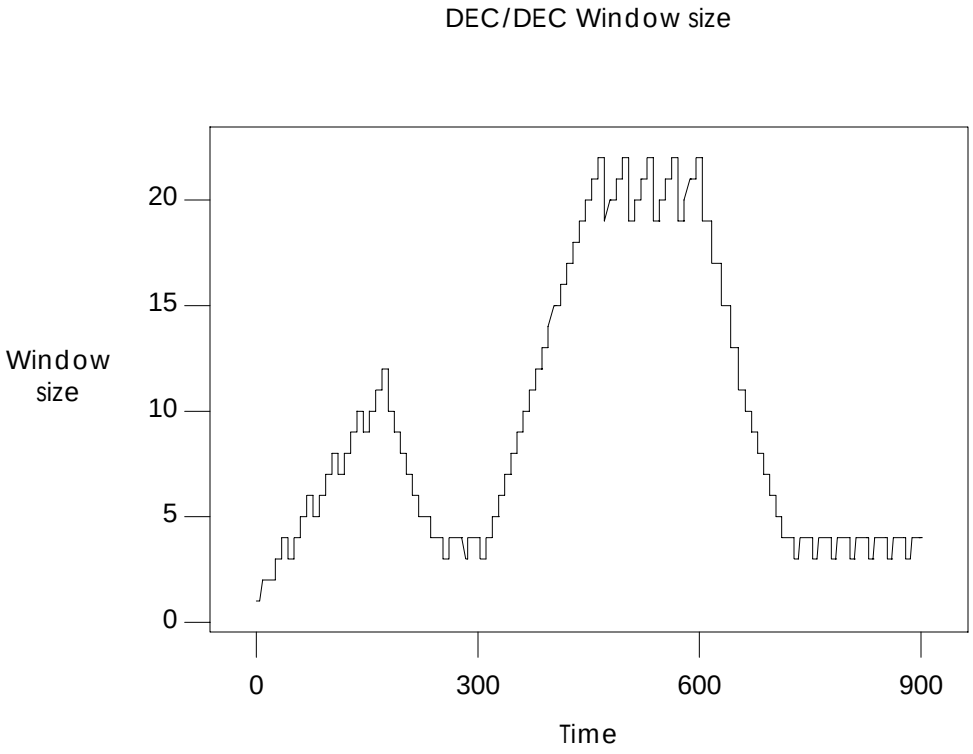


*Figure 6.11: Scenario 3: DEC/DEC Window dynamics*

When the scheduling is FQbit, the sources are protected from the background source. This has two effects. First, when the background source is active, all three sources share bandwidth equally. Second, when the background source is active, the average window size of an FTP source is larger. So, when the background source turns off, the FTP source can attain the right size more quickly, thus getting more throughput even when the background source is inactive. This is clear from Figure 6.12.

When PP_CTH flow control is used with FQ scheduling, matters are almost as good as with DECbit/FQbit, but without the need for additional information from the switches. When the background source is inactive, the two FTP sources get almost half the bottleneck bandwidth each: the bandwidth loss is because it takes a while for the sources to adjust their sending rate. When the background source is active, the three sources share the bandwidth equally. Source 2 has a few drops, since it takes a short while to react to the presence of the background source, and, in this interval, it can lose data. Source 1 does not have this problem. The inverse of the sending rate of the PP_CTH source, which corresponds roughly to the window size, is plotted in Figure 6.13.

The figure reveals that in the second period (times 600-900), source 1 window oscillates rapidly. This is because of its sensitivity to the Packet-Pair probe. In this scenario, there are only two sources, so that the consecutive probe values can differ by as much as 100%. This grossly

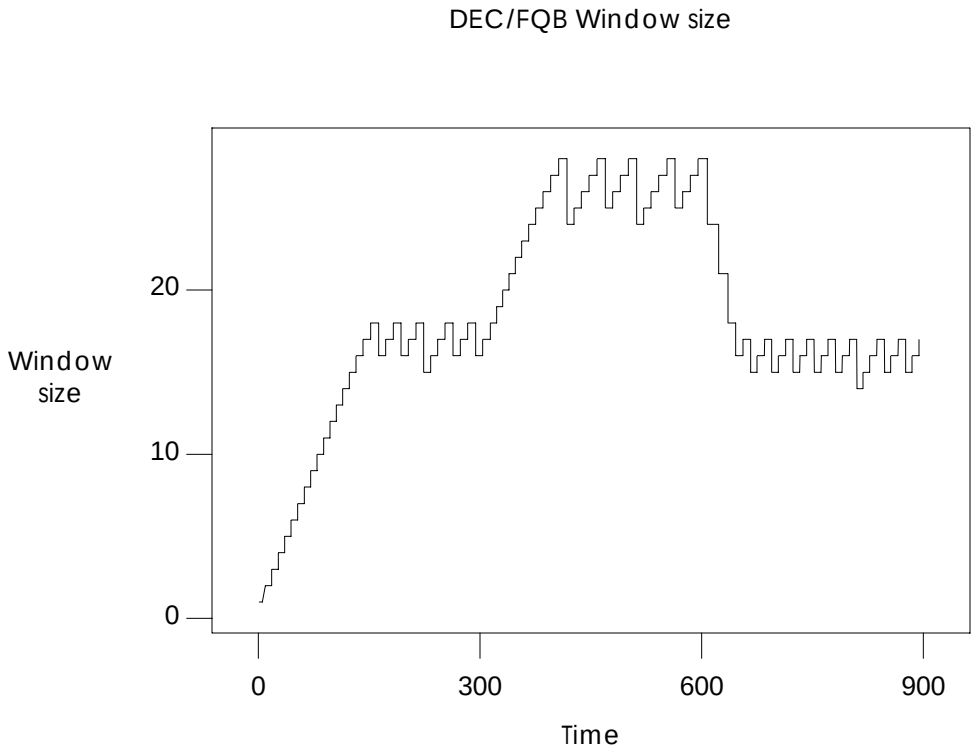


*Figure 6.12: Scenario 3: DEC/FQB Window dynamics*

violates the assumption that the system state changes somewhat slowly on a RTT time scale. Nevertheless, the overall behavior of the two sources is reasonable, as the table shows.

The other feature is a spike in the 'window' size at time 300, when the FTP source discovers the absence of the background source. 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 7.

One way to control these oscillations is to take control actions only once per 2 RTTs, as is done in the DECbit scheme. However, we believe that in high-speed networks, it is better to respond quickly, and perhaps overcompensate, than to respond too slowly, and lose throughput. This is a matter for debate and future study.

### 6.6.4. Scenario 4

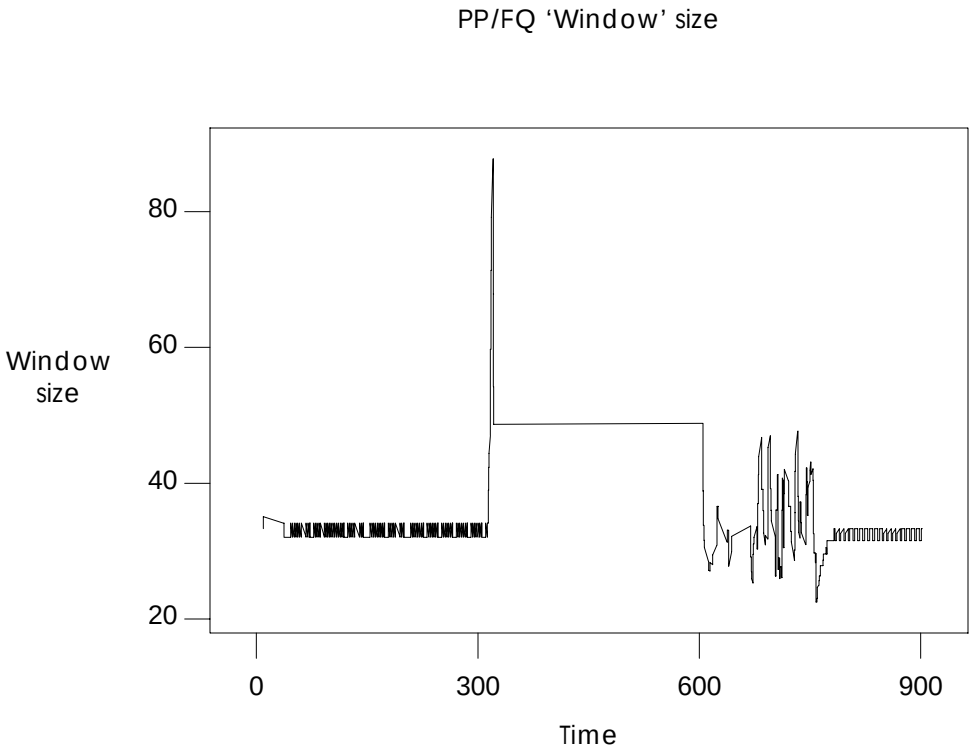


*Figure 6.13: Scenario 3: PP/FQ 'Window' dynamics*

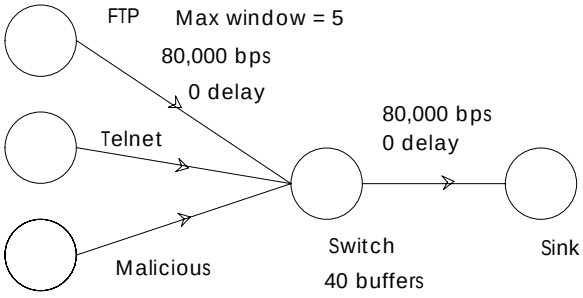


*Figure 6.14: Scenario 4*

In scenario 4 there is a single FTP and a single Telnet competing with an ill-behaved source. This ill-behaved source has no flow control and sends packets at the rate of the switch's outgoing line. This tests the ability of a scheduling algorithm to provide utility to users in the face of a malicious source. If the congestion control mechanism is poor, the FTP or Telnet source may experience a loss of utility due to the ill-behaved source.

The scenario exaggerates the effects already noted in scenario 3, and the results are summarized in Table 6.7. They are, for the most part, self explanatory. With FCFS, the FTP and Telnet sources are essentially shut out by the ill-behaved source. With FQ, they obtain their fair share of bandwidth. It is clearly seen that FCFS cannot provide any protection from malicious sources, whereas with FQ, malicious sources cannot get any more than their fair share (note that we do not punish malicious sources by incrementing the finish number even for dropped packets. Had

| Scenario 4 Results | | | | | | |
|---|---|---|---|---|---|---|
| | Throughput | | | Delay | | |
| | 1 | 2 | 3 | 1 | 2 | 3 |
| G/FCFS | 0.03 *0.07* | 9.85 *0.07* | 0 | 8.84 *2.94* | 0 | 0 |
| G/FQ | 5.03 *0.01* | 0.23 *0.05* | 4.96 *0.01* | 0.99 | 0.06 | 0 |
| JK/FCFS | 2.38 | 0 | 7.62 | 2.10 | 0 | 0 |
| JK/FQ | 5.02 | 0.21 *0.04* | 4.97 | 1.00 | 0.06 | 0 |
| DEC/DEC | 0.48 | 0 | 9.52 | 2.10 | 0 | 0 |
| DEC/FQB | 5.09 *0.01* | 0.23 *0.05* | 4.90 *0.01* | 0.73 | 0.06 | 0 |
| PP/FQ | 5.09 *0.01* | 0.23 *0.05* | 4.90 *0.01* | 0.73 | 0.06 | 0 |

| Scenario 4 Results | | | | | | |
|---|---|---|---|---|---|---|
| | Drop rate | | | Retransmission rate | | |
| | 1 | 2 | 3 | 1 | 2 | 3 |
| G/FCFS | 0.12 *0.05* | 0.15 *0.07* | 0 | 0.12 *0.04* | 0 | 0 |
| G/FQ | 0 | 0 | 5.04 *0.02* | 0 | 0 | 0 |
| JK/FCFS | 0 | 0 *0.01* | 2.38 | 0 | 0 *0.01* | 0 |
| JK/FQ | 0 | 0 | 5.03 | 0 | 0 | 0 |
| DEC/DEC | 0 | 0 | 0.48 | 0 | 0 | 0 |
| DEC/FQB | 0 | 0 | 5.10 *0.02* | 0 | 0 | 0 |
| PP/FQ | 0 | 0 | 5.10 *0.02* | 0 | 0 | 0 |

*Table 6.7: Scenario 4 simulation results*

we done so, with FQ, malicious sources would have got almost zero throughput). Thus, FQ switches are effective *firewalls* that can protect users, and the rest of the network, from being damaged by ill-behaved sources. While scenario 2 showed that FQ switches cannot control congestion by themselves, this scenario suggests that FQ switches can control the *effect* of congestion.

### 6.6.5. Scenario 5

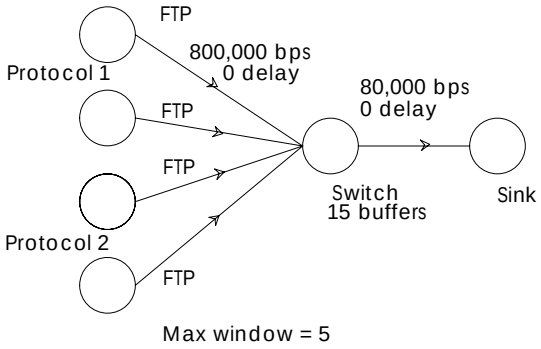


*Figure 6.15: Scenario 5*

One of the requirements of a congestion control scheme, as stated in Chapter 1, is the ability to work in heterogeneous environments. In this scenario, we test whether a congestion control algorithm can allocate bandwidth fairly with a mixture of flow control protocols at the sources. We would like the algorithm not to require a smart flow control protocol, but to provide incentives for smart ones. By comparing the relative performance of pairs of flow control

protocols for FCFS and FQ algorithms, we see how far they satisfy this criterion.

The sources have a maximum window of 5, and the buffer has a capacity of 15, hence packet losses can occur. Depending upon the flow control protocol, as well as the response of the other sources, we see a variety of outcomes. These are summarized in Table 6.8.

| JK/G with FCFS | | | | |
|---|---|---|---|---|
| | JK1 | JK2 | G1 | G2 |
| Throughput | 0.18 | 3.21 | 3.21 | 3.21 |
| Delay | 1.60 | 1.56 | 1.56 | 1.56 |
| Drop rate | 0.20 | 0 | 0 | 0 |
| Retransmission rate | 0.20 | 0 | 0 | 0 |

| JK/G with FQ | | | | |
|---|---|---|---|---|
| | JK1 | JK2 | G1 | G2 |
| Throughput | 2.75 *0.03* | 2.83 *0.03* | 2.56 *0.06* | 1.20 *0.01* |
| Delay | 1.49 *0.01* | 1.58 *0.12* | 1.34 *0.10* | 2.94 *0.03* |
| Drop rate | 0.11 | 0.06 *0.06* | 0.26 *0.08* | 0.17 |
| Retransmission rate | 0.15 | 0.07 *0.07* | 0.26 *0.08* | 0.17 |

| PP/G with FQ | | | | |
|---|---|---|---|---|
| | PP1 | PP2 | G1 | G2 |
| Throughput | 3.32 | 3.32 | 2.69 *0.01* | 0.52 *0.02* |
| Delay | 1.21 | 1.21 | 1.67 | 6.70 *0.20* |
| Drop rate | 0 | 0 | 0.07 | 0.07 |
| Retransmission rate | 0 | 0 | 0.07 | 0.07 |

| PP/JK with FQ | | | | |
|---|---|---|---|---|
| | PP1 | PP2 | JK1 | JK2 |
| Throughput | 2.50 | 2.50 | 2.50 | 2.35 |
| Delay | 1.60 | 1.20 | 2.00 | 1.16 |
| Drop rate | 0 | 0 | 0 | 0.15 |
| Retransmission rate | 0 | 0 | 0 | 0.15 |

*Table 6.8: Scenario 5 simulation results*

If the congestion control scheme were ideal, there would be no packet losses, and each source would get an equal share of the bandwidth, i.e., 2.5 packets per second. However, this is not achieved by any of the algorithm pairs.

With an FCFS switch, and Generic and JK flow controls, we find that the two Generic sources obtain a higher share of the throughput. This is because the JK sources respond to packet loss by shutting down their window, which allows the Generic sources to appropriate more than their share of the throughput. One of the two JK sources has segregated and is in the low throughput regime: this is due to the same segregation mechanism as in scenario 2.

With a FQ switch, the situation is reversed. Since each source gets an equal share of buffer space, the Generic sources, which mismanage their share, do not do as well as the JK sources. One of the two Generic sources suffers from retransmission timer backoff (indicated by the large RTT delay) and hence has a very low throughput. The JK sources also have packet losses, which is due to their congestion sensing mechanism, but overall, since they respond to the congestion signal, they perform better. Thus, the FQ switch has provided an incentive for sources to

implement JK or some other intelligent flow control, whereas the FCFS switch makes such a move sacrificial.

The other two cases judge the behavior of PP_CTH sources when they are in a heterogeneous network. The PP/G case shows the same qualitative behavior as JK/G, except that PP sources get a larger share of the throughput than the JK sources. This indicates that in simple topologies such as the one here, it does not hurt for a system to switch over from Generic to PP.

With JK and PP sources, all four sources get an equal share of the raw throughput. However, since source 4 has a retransmission rate of 0.15, its effective throughput goes down to 2.35 packets/sec. One of the PP sources has a larger RTT delay: this is because of the error in the first probe, as explained for scenario 1.

### 6.6.6. Scenario 6

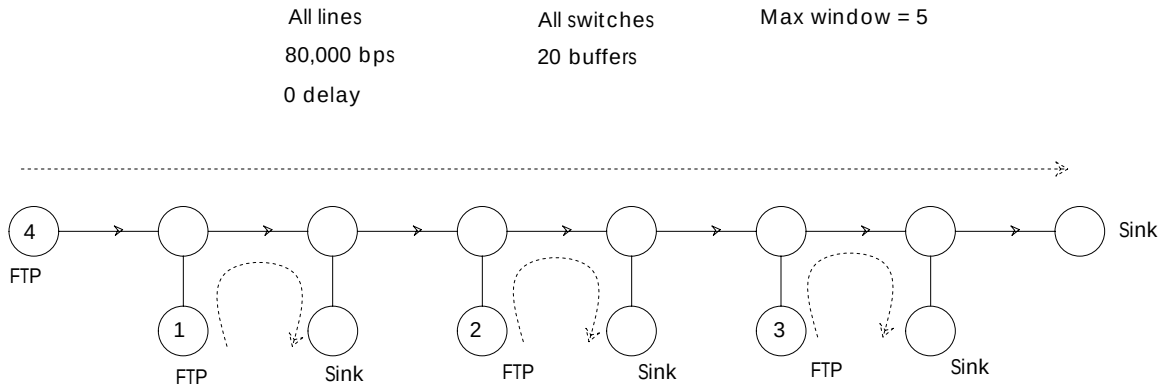


*Figure 6.16: Scenario 6*

We noted earlier that a window-based scheme tends to allocate bandwidth unfairly - conversations spanning a shorter number of hops will get a larger bandwidth allocation than conversations over longer paths (unless congestion signals are selectively set). Scenario 6 has a multinode network with four FTP sources using different network paths. Three of the sources have short mutually nonoverlapping conversations and the fourth source has a long path that intersects each of the short paths. The simulation results are presented in Table 6.9. There were no packet losses or retransmissions, so these results are omitted.

In the ideal case, both the long and short conversations should share the bottlenecks, so that their throughputs should be 5 packets/s each. The G/FCFS protocol pair gives higher throughput to the shorter conversation, as expected. Shorter conversations get acks back, and can send the next packet out, sooner than the long conversation. Hence, they get higher throughput. With FQ, the bandwidth received by a source is no longer inversely proportional to its RTT delay; so both long and short conversations receive the same throughput. Almost identical results hold for JK sources, and for the same reasons.

With DEC sources, the exact division of throughput depends in detail on the choice of a parameter called the *capacity factor*, on whether or not the switch can set bits in panic mode, and on the maximum size of the window. With varying parameters, nearly fair, as well as grossly unfair, throughput distributions are possible. We present results with a choice of capacity factor of 0.9, maximum window size of 5, and the switch allowed to go into panic mode. In this situation, around 12% of the bandwidth is wasted, and the short conversations get about 20% more than their fair share of the bottleneck bandwidth. The long conversation gets much less, but, as we mentioned, with a different choice of parameters, it can get as much as 80% of its fair share.

Such sensitivity to parameters, as well as unfair bandwidth allocation, vanishes with the FQbit scheme. The results for PP_CTH/FQ are also nearly ideal.

| Scenario 6: Throughput | | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| G/FCFS | 7.14 | 7.14 | 7.14 | 2.86 |
| G/FQ | 5.29 *0.21* | 5.29 *0.21* | 5.29 *0.21* | 4.71 *0.21* |
| JK/FCFS | 7.14 | 7.14 | 7.14 | 2.86 |
| JK/FQ | 5.00 | 5.00 | 5.00 | 5.00 |
| DEC/DEC | 6.08 *0.04* | 6.14 *0.06* | 6.15 *0.05* | 2.67 *0.07* |
| DEC/FQB | 5.00 | 5.00 | 5.00 | 5.00 |
| PP/FQ | 5.00 | 5.00 | 5.00 | 5.00 |

| Scenario 6: Delay | | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| G/FCFS | 0.70 | 0.70 | 0.70 | 1.75 |
| G/FQ | 0.95 *0.04* | 0.95 *0.04* | 0.95 *0.04* | 1.06 *0.04* |
| JK/FCFS | 0.70 | 0.70 | 0.70 | 1.75 |
| JK/FQ | 1.00 | 1.00 | 1.00 | 1.00 |
| DEC/DEC | 0.37 | 0.37 | 0.38 | 0.88 |
| DEC/FQB | 0.91 | 0.91 | 0.91 | 1.00 |
| PP/FQ | 1.00 | 1.00 | 1.00 | 1.00 |

*Table 6.9: Scenario 6 simulation results*

### 6.6.7. Scenario 7

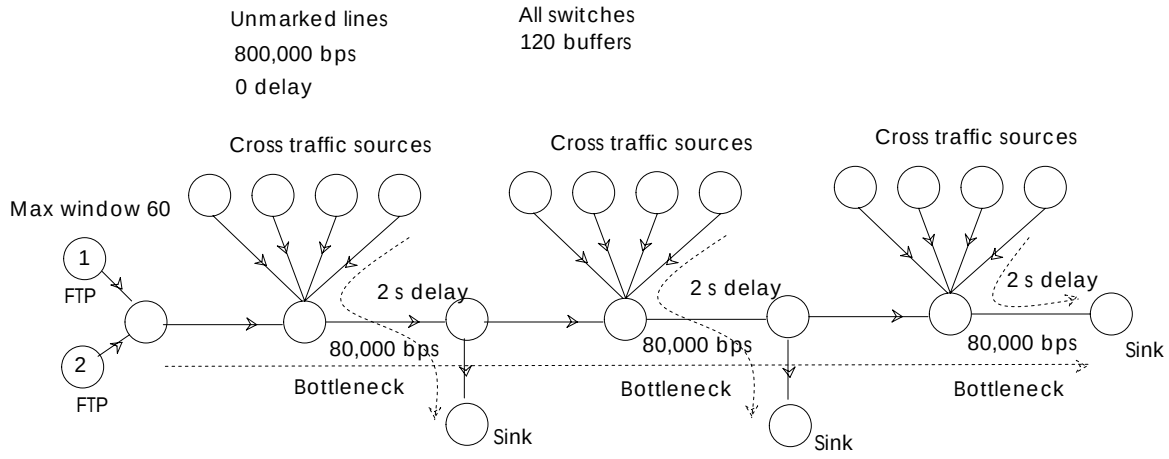


*Figure 6.17: Scenario 7*

Scenarios 1-6 have explored the behavior of the FQ algorithm and PP flow control in networks 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. In Chapter 5, we presented a stochastic model for a conversation in a network with dynamically varying state, and proposed mechanisms that allow sources to respond to changes in this state. The performance of these mechanisms is explored in scenarios 7 and 8, and is compared to the performance of some other schemes.

Scenario 7 (Figure 6.17) explores problems that arise when there are large propagation delays, as well as three potential bottlenecks created by cross traffic from Poisson sources (though a delay of 2 s 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). 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 observation noise.

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 [54]. Here, the maximum 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 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.

The simulation results are summarized in Table 6.10.

| Scenario 7: 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* |
| DEC/DEC | 0.08 | 0.10 *0.03* | 12.97 *0.07* | 12.97 *0.10* |
| DEC/FQB | 1.65 *0.15* | 1.65 *0.15* | 15.22 *0.93* | 15.22 *0.93* |
| PP/FQ | 1.70 *0.02* | 1.72 *0.01* | 16.64 *4.97* | 19.58 *4.37* |

| Scenario 7: Drop rate and retransmission rate | | | | |
|---|---|---|---|---|
| | Drop 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 |
| DEC/DEC | 0 | 0 | 0 | 0 |
| DEC/FQB | 0 | 0 | 0 | 0 |
| PP/FQ | 0 | 0 | 0 | 0 |

*Table 6.10: Scenario 7: simulation results*

The Poisson sources in this scenario are 'ill-behaved'; so, as expected, neither the Generic sources, nor the DEC/DEC pair does well in this scenario. Since the reasons for this have been examined earlier, we will only concentrate on the other four 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 6.18.

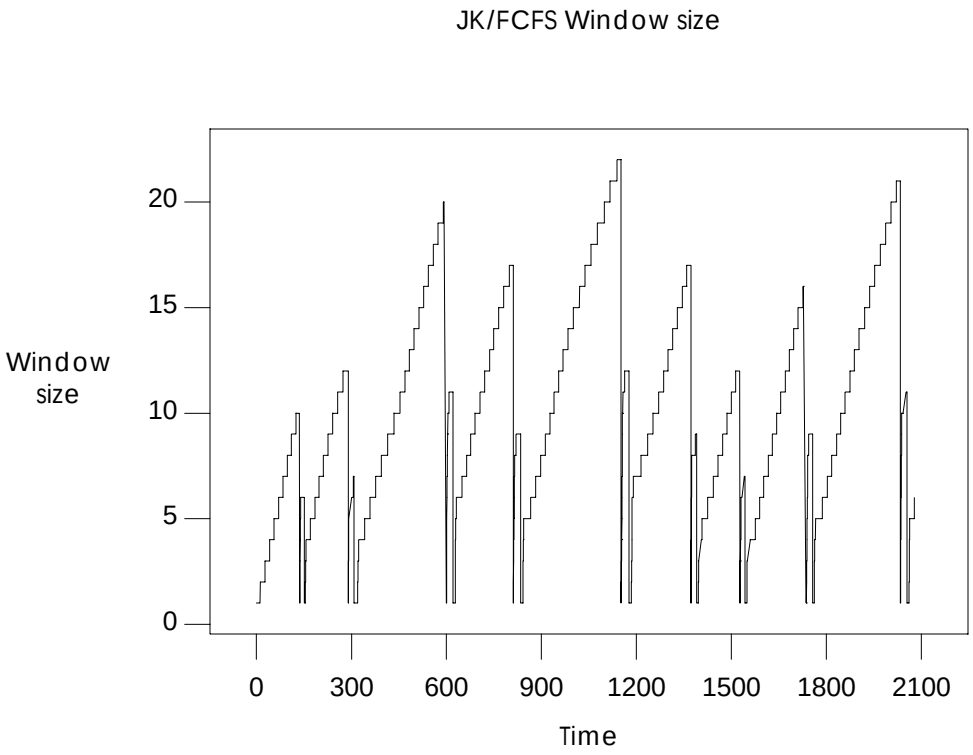


*Figure 6.18: Scenario 7: JK/FCFS window vs. time*

Note that the highest window size achieved is around 25, 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).

When the scheduling discipline is changed to FQ, the situation improves considerably (Figure 6.19). The maximum window achieved is around 35, 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, that is adversely affected by each shutdown, rather than to FQ.

The DEC/DEC protocol pair performs poorly: the ill-behaved Poisson sources force the FTP sources to shut their window down, and the window size never goes beyond 2. When FQbit provides protection, though, the two sources achieve almost their entire fair share of the bandwidth (with a much lower queueing delay than JK/FQ). The window vs. time diagram for source 1 is presented in Figure 6.20. Note that, though the source takes a while to reach the steady state, once it is there, it proceeds to send data relatively undisturbed by the Poisson sources.

The PP_CTH protocol can respond rapidly to changes in network state. Thus, if any of the Poisson sources is idle, the PP_CTH source can detect this, and make use of the idle time. Hence, the two PP_CTH sources obtain more than their fair share of the throughput (Table 6.10).

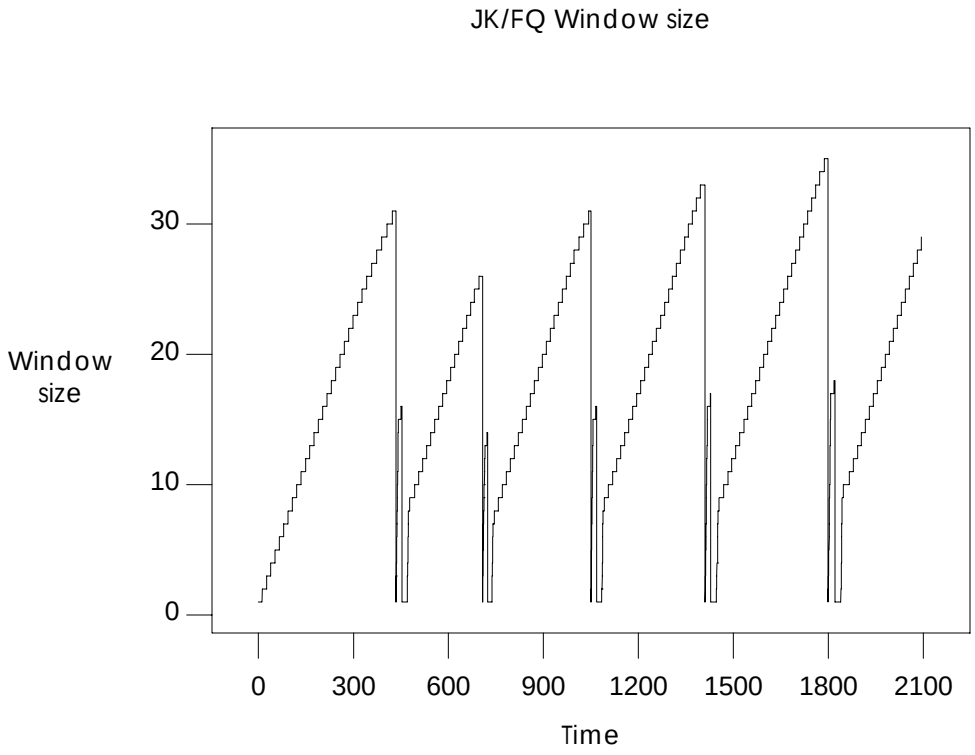


*Figure 6.19: Scenario 7: JK/FQ window vs. time*

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. This vindicates our decision to use a control-theoretic basis to design flow control mechanisms, since we outperform even the DECbit algorithm, but without using explicit congestion signals, in a scenario that challenges many of the simplifying assumptions made in Chapter 5 (that is, that the number of sources in the cross traffic is large, that the observation noise variance is small, that bottlenecks do not migrate, as well as numerous less important assumptions). We now examine the performance of the protocol in some more detail.

Figure 6.21 shows the typical inter-acknowledgement spacing as seen by source 1, and its corresponding choice of inter-packet spacing (which is the inverse of the sending rate or 'window' size), over a period of 100 seconds. The square-wave like line represents the probe value, and the other solid line represents the inter-packet spacing of the packets being transmitted from the PP_CTH source. The dotted line shows the value of the exponential averaging constant $\alpha$.

The inter-ack spacing depends on how many other packets get service between two packets from source 1. Since each packet takes 0.1 seconds to get service, the interpacket spacing has to be 0.3, 0.4, 0.5 etc., hence the rectangular pattern. The inter-packet spacing, for the most part, tracks the inter-ack spacing. Note that the source ignores single spikes in the input, and that, whenever the probe values stabilize, the sending rate catches up to the probe value exponentially, as explained in Chapter 5. The fact that the source tracks the input rather closely shows the effectiveness of the fuzzy controller. Essentially, the controller drops the value of $\alpha$ whenever the prediction error is large. This allows it to quickly catch up with the probe value. Since the estimator for the prediction error ignores spikes, a large error is very likely due to the start or the end of a conversation, and, adapting to these changes, the source is able to

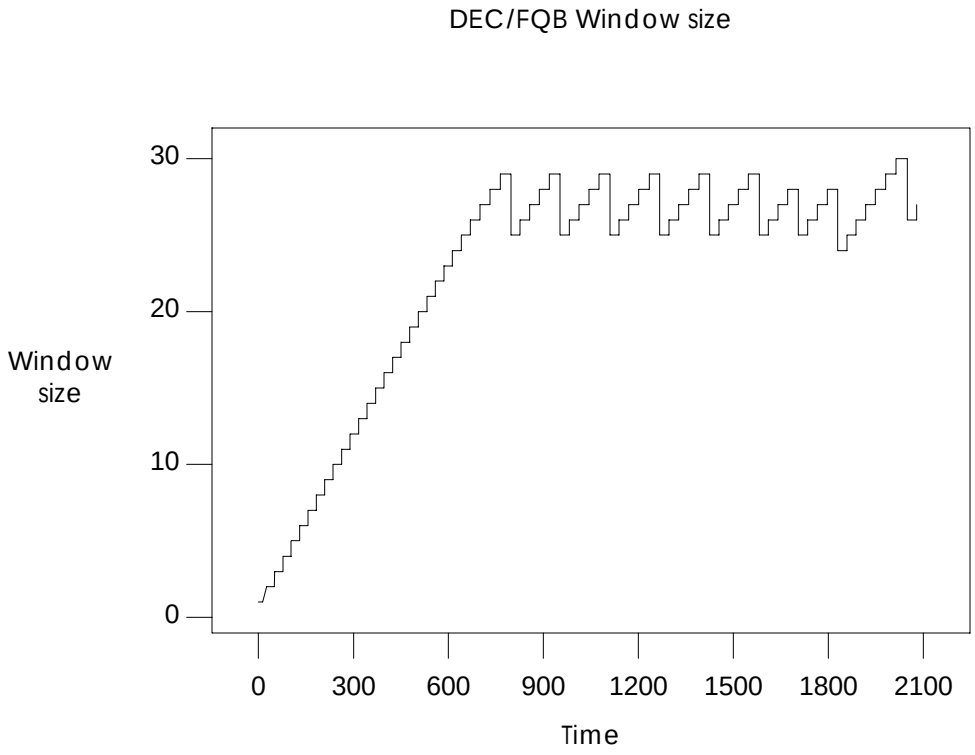


*Figure 6.20: Scenario 7: DEC/FQB window vs. time*

send more data than it could otherwise.

We had earlier cautioned that it is necessary to do 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 (Figure 6.22).

The figure shows that the number of outstanding packets shoots up rapidly, stops at 60, 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 60, 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 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.

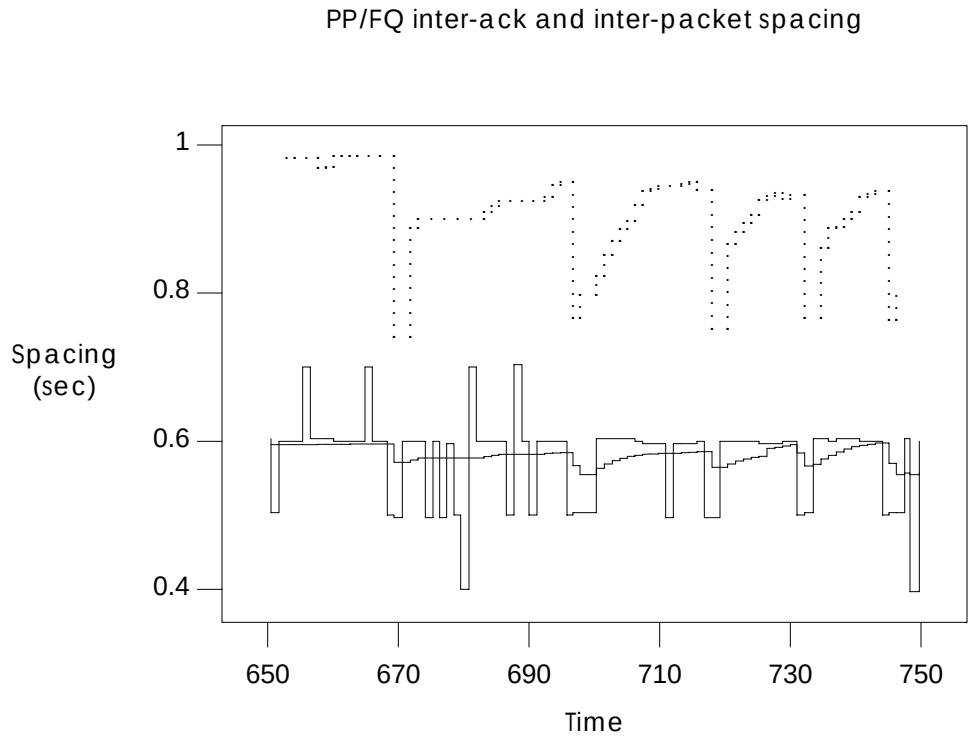


*Figure 6.21: Scenario 7: PP/FQ inter-ack spacing, inter-packet spacing and α*

### 6.6.8. Scenario 8

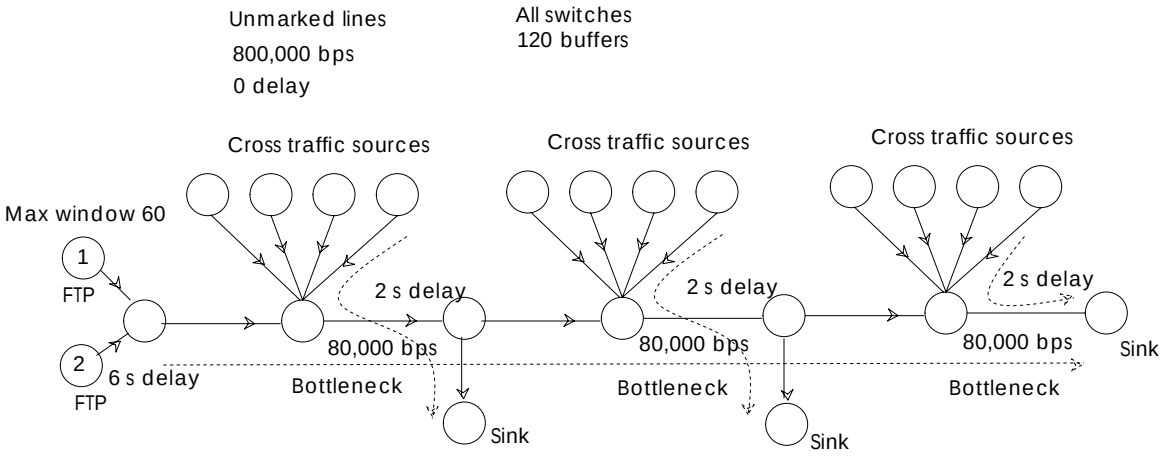


*Figure 6.23: Scenario 8*

Scenario 8 is similar to scenario 7, except that source 1 has a round-trip-time delay of 12 seconds, and source 2, of 24 seconds (see Figure 6.23). 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 6.11.

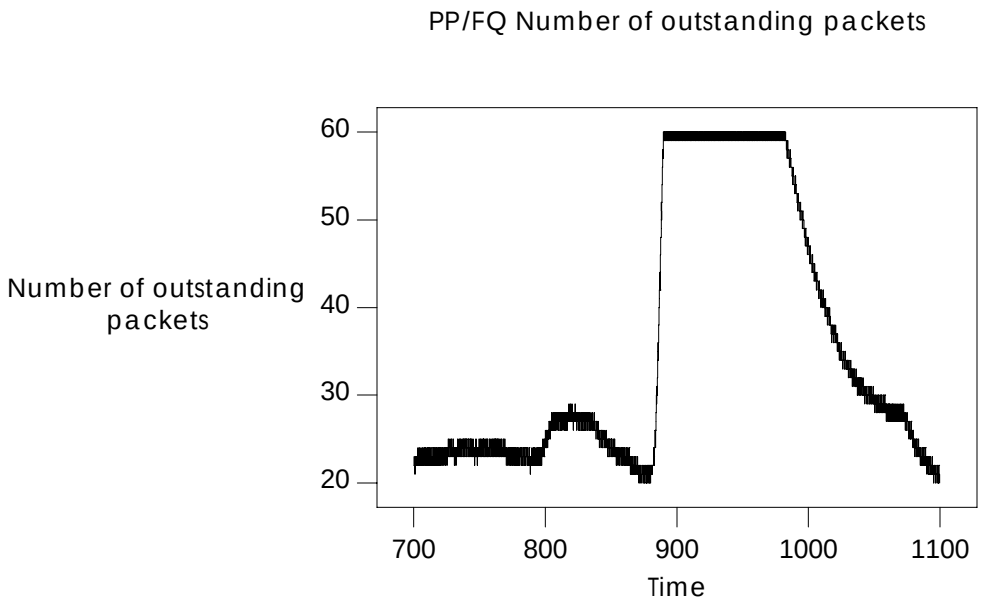


*Figure 6.22: Scenario 7: Number of outstanding packets vs. time*

| Scenario 8: 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* |
| DEC/DEC | 0.09 *0.01* | 0.03 | 12.96 *0.06* | 12.32 *0.06* |
| DEC/FQB | 1.90 *0.23* | 0.03 | 16.73 *2.30* | 12.30 *0.11* |
| PP/FQ | 1.73 *0.02* | 1.64 | 22.11 *3.48* | 36.50 |

| Scenario 8: Drop rate and retransmission rate | | | | |
|---|---|---|---|---|
| | Drop 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 |
| DEC/DEC | 0 | 0 | 0 | 0.03 |
| DEC/FQB | 0 | 0 | 0 | 0.03 |
| PP/FQ | 0 | 0 | 0 | 0 |

*Table 6.11: Scenario 8 simulation results*

As in scenario 7, 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 DEC/DEC pair is shut out, as before. 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. Source 2 of the DEC/FQbit pair loses throughput for exactly the same reason. (This effect was not present in scenario 6, where there were no packet losses.)

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_CTH behaves well even under fairly adverse conditions.

## 6.7. Conclusions

In the previous sections, we have presented and justified our simulation methodology, and have presented detailed simulation results for a suite of eight benchmarks. In this section, we summarize our conclusions.

Our overall results are encouraging. We have shown that FQ does better than FCFS in almost all the scenarios. Further, the PP flow control protocol consistently matches or outperforms the competing JK and selective DECbit schemes. Thus, we claim that our venture to design efficient and robust congestion control schemes has been successful. We justify this claim by reviewing the results from the simulations.

Scenario 1 showed that, unlike FCFS, FQ can provide lower delays, and hence more utility, to delay-sensitive Telnet sources. All the protocol pairs work well here, but as we examine the other scenarios, they exhibit their weaknesses. Thus, we should be cautious of simulation results for an 'average' case: the average case may hide poor performance under adverse conditions.

We also saw that the queueing delay incurred by the packets generated by a PP_CTH FTP source depends somewhat on the initial estimate of the propagation delay. Since FTP sources do not lose utility from increased delay, this sensitivity does not pose problems in this case. Note that, if one of the sources somehow obtained a completely wrong estimate of the propagation delay, then it would accidentally choose a very low setpoint and lose throughput, leading to a loss of utility. However, the error would arise mainly from phase delay, and, as we argued in §5.7, this can only slightly affect the estimate. So, we do not expect FTPs to lose utility due to an incorrect estimate of the propagation delay. The argument is also justified by our simulations, where competing PP_CTH sources, some of which have errors in their propagation delay estimates, obtain nearly identical throughputs and delays.

We earlier claimed that the role of a congestion control scheme is to provide utility to the users of the network. FQ allows users to choose their throughput delay tradeoff by choosing their own setpoint for the size of the bottleneck queue. FCFS links the throughput and delay allocations, and makes individual tradeoffs impossible (though global tradeoffs are still possible [6]). The DEC algorithm controls the queueing delay by attempting to keep the average queue size close to one. However, it does not allow individual users to make different delay/throughput tradeoffs; the collective tradeoff is set by the switch.

Scenario 2 examined the situation when congestion could occur due to the overloading of a switch by 6 FTP sources. We found that the Generic flow control algorithm is insensitive to congestion, and hence performs poorly with both the FCFS and FQ scheduling disciplines. To get utility, not only must the network provide an appropriate scheduling discipline, the sources must also react intelligently to network state changes. The JK, DEC and PP_CTH protocols behave well in this scenario, since they respond adequately to congestion signals.

Scenario 2 also was the first one to exhibit segregation of the sources. The origin of segregation for FCFS sources is easily understood, and has also been studied exhaustively in references [39, 40]. The slight segregation with FQ is harder to explain, and depends in detail on the

exact implementation of the JK source. The major conclusion to draw from this is that segregation can arise from a number of sources, but, once it arises, it is self sustaining. However, it must be borne in mind that segregation is sensitive to the exact values of link speeds, buffer capacities and maximum window sizes, and for a wide range of these values, segregation does not occur.

Scenario 3 examined the dynamic behavior of flow control algorithms in response to an abrupt change in the network state. We saw that both JK and DEC take a while to respond to the change, while PP responds immediately. This is the reason why, in scenario 8, PP outperforms the other schemes.

The firewall property of FQ is graphically demonstrated in scenario 4. Here, malicious or ill-behaved sources can completely disrupt the flow of other sources with FCFS switches, while FQ switches prevent such abuse.

The four cases of scenario 5 demonstrate two things. First, FQ provides an incentive for users to use an intelligent flow control protocol, such as PP_CTH or JK. In contrast, FCFS makes such a move sacrificial. Second, PP_CTH works well in networks that have combinations of PP_CTH and both JK and Generic sources. Users using PP_CTH can only benefit from using it, which is an incentive to convert to it.

Scenario 6 shows that unlike FCFS, FQ does not discriminate against conversations with long paths. This is important as the scale of WANs increases.

Scenarios 7 and 8 try to challenge the assumptions made in the design of PP_CTH, and test the robustness of the algorithm in adverse conditions. In Chapter 5, we assumed that the bottleneck service rate does not rapidly fluctuate - in scenario 7, on the contrary, Figure 6.21 shows that the inter-ack spacing probe fluctuates rapidly. Further, we had assumed that the fluctuations in the probe value would be airly small, whereas the changes in the probe value in Scenario 7 are as large as 10% and 30%. Third, there are three bottlenecks in tandem, so that bottleneck migration is possible, and can lead to observation noise. Fourth, there are no buffer reservations, as is recommended when using PP_CTH. Finally, the PP_CTH source has a long propagation delay, so that the probe values report stale data. In spite of these difficulties, PP_CTH behaves rather well. This gives us confidence in our design methodology.

The adverse conditions of scenario 7 are worsened in scenario 8, where one source has double the propagation delay of the other. We see that only PP_CTH is able to deliver reasonably fair throughput to the two sources in this scenario.

To summarize, we have shown that both FQ and PP_CTH work well as congestion control mechanisms, and they work well together. This conclusion is valid to the extent that our suite of benchmarks is comprehensive. While we have tested for ill-behaved users, severe buffer contention, the differing utilities of FTP and Telnet sources, 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 worst-case 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 our schemes have several promising features that may make them suitable for future high speed networks. Studying their effectiveness in the real world is an area for much future work.