Congestion Control in Computer Networks

By

Srinivasan Keshav

B.Tech. (Indian Institute of Technology, Delhi) 1986
M.S. (University of California) 1988

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA at BERKELEY

Approved:

Chair: .... August 23, 1991..
Date

Pravin Varaiya .... June 19 '91

.... 8 August 1991

* *

Congestion Control in Computer Networks

by

Srinivasan Keshav

Congestion Control in Computer Networks

by

Srinivasan Keshav

Abstract

This thesis examines the problem of congestion control in reservationless packet switched wide area data networks. We define congestion as the loss of utility to a network user due to high traffic loads and congestion control mechanisms as those that maximize a user's utility at high traffic loads. In this thesis, we study mechanisms that act at two time scales: multiple round trip times and less than one round trip time. At these time scales, congestion control involves the scheduling discipline at the output trunks of switches and routers, and the flow control protocol at the transport layer of the hosts.

We initially consider the problem of protecting well-behaved users from congestion caused by ill-behaved users by allocating all users a fair share of the network bandwidth. This motivates the design and analysis of the Fair Queueing resource scheduling discipline. We then study the efficient implementation of the discipline by doing an average case performance evaluation of several data structures for packet buffering.

Since a Fair Queueing server maintains logically separate per-conversation queues and approximates a bitwise-round robin server, it partially decouples the service received by incoming traffic streams. This allows us to deterministically model a single conversation in a network of Fair Queueing servers. Analysis of the model shows that a source can estimate the service rate of the slowest server in the path to its destination (the bottleneck) by sending a pair of back-to-back packets (a packet-pair probe), and measuring the inter-acknowledgement spacing. The probe values can be used to control a users's data sending rate. We formalize this notion by modeling a conversation as a linear system in a simple control-theoretic approach. This is used to synthesize a robust and provably stable flow control protocol. The network state, that is, the service rate of the bottleneck, can be estimated from the series of probe values using an estimator based on elementary fuzzy logic.

Our analysis and performance claims are examined by simulation experiments on a set of eight test scenarios. We show that under a wide variety of test conditions, both of our schemes provide users with good performance. Thus, these mechanisms should prove useful in future high-speed networks.

Table of Contents

List of Tables

List of Figures

Acknowledgements

My first debt is to my advisor, Prof. Ferrari, who gave me a free hand with my research, and asked all the right questions. I have been greatly influenced by his tenet of bridging the gap between theory and practice, and have benefited from his detailed comments on every aspect of this thesis.

My other mentor is Scott Shenker, who introduced me to the area of congestion control. His style of research and emphasis on intellectual rigor will always be a guiding influence.

I would like to thank Sam Morgan, Chuck Kalmanek, Hemant Kanakia and Ellen Hahne at AT&T Bell Laboratories for providing a stimulating research environment during my visits, and to Sandy Fraser for the research funding that made much of this work possible.

Chapter 2, and a part of Chapter 6, present joint work with Scott Shenker and Alan Demers at Xerox PARC. I gratefully acknowledge their guidance and permission to use this work in the thesis. The use of deterministic analysis in Chapter 4 was motivated by work done by Prof. Samar Singh at the Indian Institute of Technology, Delhi, and Prof. Ashok Agrawala at the University of Maryland at College Park. They also independently invented the packet-pair probing technique. In Chapter 5, the use of a linear system model was suggested by G. Srinivasan. Pratap Khedkar gave valuable advice on the fuzzy controller. The helpful comments and criticism of Dr. R.P. Singh, Dr. Debasis Mitra, Prof. M. Tomizuka and Prof. P.P. Varaiya is much appreciated. The REAL simulator used in Chapter 6 is based on the NEST simulation package from Columbia University. My thanks to Alex Dupuy who helped me through some hairy debugging sessions, and to Steve McCanne who made several improvements and bug fixes to REAL.

Last, but not the least, my thanks to Dinesh Verma, on whom I could count for helpful criticism at a moments notice, and to my present and former colleagues, for their help, advice, and stimulating discussions: Anindo Banerjea, Ramon Caceres, Peter Danzig, Sally Floyd, Riccardo Gusella, Pratap Khedkar, Sharad Malik, Dave Presotto, Venkat Rangan, Stuart Sechrest, G. Srinivasan, Mark Sullivan, Shin-Yuan Tzou, Bernd Wolfinger and Hui Zhang,

Chapter 3 is being concurrently published as ‘‘On the Efficient Implementation of Fair Queueing’’ in the Journal of Internetworking: Research and Experience, and is reprinted by permission of John Wiley and Sons, Ltd.

Chapter 1: Introduction

1.1. Historical perspective

Computer networks form an essential substrate for a variety of distributed applications, but they are expensive to build and operate. This makes it important to optimize their performance so that users can derive the most benefit at the least cost. Though most networks perform well when lightly used, problems can appear when the network load increases. Loosely speaking, congestion refers to a loss of network performance when a network is heavily loaded. Since congestive phenomena can cause data loss, large delays in data transmission, and a large variance in these delays, controlling or avoiding congestion is a critical problem in network management and design. This dissertation presents some approaches for congestion control in wide-area computer networks.

Historically, the first wide-area networks (WANs) were circuit-switched telephone networks. Since these networks carry traffic of a single type, and the traffic behavior is well known, it is possible to avoid congestion simply by reserving enough resources at the start of each call. By limiting the total number of users, each admitted call can be guaranteed to have enough resources to achieve its performance target, and so there is no congestion. However, resources can be severely underutilized, since the resources blocked by a call, even if idle, are not available to other calls.

Early research in computer data networking led to the development of reservationless store-and-forward data networks [132]. These networks are prone to congestion since neither the number of users nor their workload are regulated. Essentially, the efficiency gained by statistical multiplexing of network resources is traded off with the possibility of congestion. This problem was recognized quite early [22], and a number of congestion control schemes were proposed; references [44, 110] provide a detailed review of these.

In the past three years, there has been a renewed interest in congestion control, as a glance at the Bibliography will indicate. We feel that at least three factors have been responsible. First, the spread of networks such as ARPANET, NSFNET, CSNET and BITNET, and their interconnection, has created a very large Internet whose size has made it unmanageable. The large number of users and a complete decentralization of network management made it inevitable that congestion would pose problems sooner or later. Matters came to a crisis around mid-1987, and prompted two pioneering research efforts. Jain, Ramakrishnan and Chiu at Digital Equipment Corporation developed the DECbit congestion control scheme [117]. Simultaneously, Jacobson at Lawrence Berkeley Laboratories and Karels at UC Berkeley modified the well-known Transmission Control Protocol (TCP) [21, 108] to intelligently react to congestion, and to recover from it [63]. The success of these efforts brought congestion control into focus as a major research area in the Internet community.

The second factor is the development of optical fiber technology. Optical fiber trunks offer data bandwidths that are a factor of 10,000 larger than earlier circuits (a ratio of 600 Mbps to 56kbps). A number of researchers have recognized the critical role of congestion control in high-speed WANs, since the bandwidth-delay product of a single circuit in such networks can be as large as 30 Mbits (600 Mbps x 50 ms round-trip delay across the USA). With such large products, a *single* source could introduce a transient load large enough to swamp buffers at the switches, leading to packet losses and excessive end-to-end delays for *all* the hosts on the network.

The third factor is social, rather than technological. The networking community has long been divided into two camps: the computer data networking community, and the telecommunications community. However, in recent years, the telecom community has realized the benefits of packet switching, resulting in the Asynchronous Transmission Mode (ATM) proposal from CCITT. Similarly, data networking researchers have realized that they need to provide real-time bounds on data transfer for services such CD-quality audio and interactive video [35, 154]. This growing together of the two communities has led to a cross-fertilization of ideas about congestion management. Indeed, some of the most exciting proposals for future networks are arising from this interaction.

The present time appears to be critical for the design of future high speed networks, and in particular, their congestion control mechanisms. In this dissertation, we propose a number of ideas that we believe are useful for high speed networks. We hope that our work will contribute to the ongoing debate about congestion control.

This chapter is laid out as follows. Section 1.2 presents the environment of discourse. Section 1.3 defines congestion, and section 1.4 defines and discusses congestion control. Section 1.5 describes the fundamental requirements of a congestion control scheme, and the next section presents the fundamental assumptions made in this thesis. Section 1.7 reviews previous work, and the scope of this thesis is presented in section 1.8.

1.2. Environment of discourse

We survey congestion control techniques in two types of wide-area networks (though the dissertation is limited to techniques suitable for the first type). In both networks, data is sent from *sources* of data to *sinks* through intermediate store-and-forward switching nodes. Sources of data could be human users, transferring characters in a remote login session, or transferring files. For our purposes, we will refer to processes at OSI layer five and above as data sources. Sinks are the ultimate destinations of the data. They are the peer processes of the sources that receive and consume the received data, and they are typically assumed to acknowledge the receipt of each packet. *Switches* route and schedule incoming packets on outgoing lines, placing data in *output buffers* when the arrival rate exceeds the service rate. The simplex stream of packets between a source of data and its sink is called a *conversation*. Usually, a conversation corresponds to a pair of transport level endpoints, for example, two BSD sockets [21, 23].

The first type of network under consideration, called a *reservationless network*, is an abstract model for networks such as the Internet. In such a network, while intermediate switches may reserve buffers (which does not reduce statistical multiplexing of the bandwidth), they may not reserve bandwidth (which does). Hosts on reservationless networks are assumed to be connected directly to switches, that in turn connect to other switches or hosts. A switch could be a piece of software that resides in a host, or could be a separate piece of hardware.

The other type of networks are those where switches reserve both bandwidth and buffers on behalf of *Virtual Circuits* (VCs) (such as in Datakit [41]). We call these *reservation-oriented networks*. We assume that these networks carry two types of traffic: performance-oriented traffic, which usually needs some form of real-time delay, bandwidth and jitter guarantees, and best-effort data traffic, which does not make such demands [35]. Since no bandwidth is reserved on behalf of best-effort traffic [68], the best-effort component of a reservation-oriented network can be modeled as a reservationless network. Hence, schemes that are designed for reservationless networks can be transferred, with appropriate modifications, to reservation-oriented networks.

We believe that most future generation networks will tend to be reservation-oriented. Nevertheless, there are still some valid reasons to study congestion control in reservationless networks. First, reservationless networks will always be able to use bandwidth more efficiently than reservation-oriented networks due to the gain from statistical multiplexing. So, network providers who want to optimize cost will continue to build reservationless networks. Second, the techniques that are developed for congestion control can be applied to control best-effort traffic in reservation-oriented networks. Thus, the results of this work will apply even in those networks. Third, reservationless networks are currently the most common type of computer network. We believe that because of inertia, and a desire to stay with known and proven technology, they will continue to exist in the future.

1.3. What is congestion?

Despite the large and rapidly proliferating literature on congestion control, we are not aware of a satisfactory definition of congestion. We now discuss some common definitions, point out their flaws, and then propose a new definition that we consider to be superior.

Some common definitions of congestion

Since congestion occurs at high network loads, definitions of congestion focus on some aspect of network behavior under high load. We first discuss a scenario that leads to network congestion in reservationless networks, and then motivate some definitions.

Consider a reservationless network, where, due to some reason, the short term packet arrival rate at some switch exceeds its service rate. (The service rate is determined by the processing time per packet and the bandwidth of the output line. Thus, the bottleneck could be either the switch's CPU or the outgoing line: in either case, there is congestion.) At this point, packets are buffered, leading to delays. The additional delay can cause sources to time out and retransmit, increasing the load on the bottleneck [101]. This feedback leads to a rapidly deteriorating situation where retransmissions dominate the traffic, and effective throughput rapidly diminishes [63,114]. Further, if there is switch to switch flow control (as in ARPANET, Tymnet, Datapac, Sirpent, etc.), new packets may not be allowed to enter the switch, and so packets might be delayed at a preceding switch as well. This can lead to deadlock, where all traffic comes to a standstill [132].

Note that three things happen simultaneously. First, the queueing delay of the data packets increases. Second, there may be packet losses. Finally, in the congested state, the traffic is dominated by retransmissions, so that the effective data rate *decreases*. The standard definitions of congestion are thus of the form: ''A network is congested if, due to overload, condition X occurs'', where X is excessive queueing delay, packet loss or decrease in effective throughput. The first definition is used in references [66,117], the second in reference [63], and the third in reference [85].

These definitions are not satisfactory for several reasons. First, delays and losses are indices of performance that are being improperly used as indices of congestion, since the change in the indices may be due to symptoms of phenomena other than congestion. Second, the definitions do not specify the exact point at which the network can be said to be congested (except in a deterministic network, where the knee of the load-delay curve, and hence congestion, is well defined, but that is the trivial case). For example, while a network that has mean queueing delays in each switch of the order of 1 to 10 service times is certainly not congested, it is not clear whether a network that has a queueing delay of 1000 service times is congested or not. It does not seem possible to come up with any reasonable threshold value to determine congestion!

Third, a network that is congested from the perspective of one user is not necessarily congested from the perspective of another. For example, if user A can tolerate a packet loss rate of 1 in 1000, and user B can tolerate a packet loss rate of 1 in 100, and the actual loss rate is 1 in 500, then A will claim that the network is congested, whereas B will not. A network should be called uncongested only if all the users agree that it is.

New definition

From the discussion above, it is clear that network congestion depends on a user's perspective. A user who demands little from the network can tolerate a loss in performance much better than a more demanding user. For example, a user who uses a network only to send and receive electronic mail will be happy with a delivery delay of a day, while this performance is unacceptable for a user who uses a network for real-time audio communication. The key point is the notion of the *utility* that a user gets from the network, and how this utility degrades with network loading.

The concept of ‘utility’ used here is borrowed from economic theory. It is used to refer to a user’s preference for a resource, or a set of resources (often called a resource bundle). Strictly speaking, the utility of a user is a number that represents the relative preference of that user for a resource (or performance) bundle, so that, if a user prefers bundle A to bundle B, the utility of A is greater than the utility of B. For example, if A is {end-to-end delay of 1 second, average throughput 200 pkts/second}, and B is {end-to-end delay of 100 seconds, average throughput 20000 pkts/second}, a user may prefer A to B, and we would assign a utility to A that is greater than the utility of B, while another user may do the opposite. In classic microeconomic theory, utilities are represented by a function over the resources [140]. Since utilities express only a preference ordering, utility functions are insensitive to monotonic translations, and the utilities of two users cannot be compared; the function can only be used to relatively rank two resource bundles from the point of view of a single user.

An example of a utility function is $\alpha T - (1-\alpha)RTT$, where α is a weighting constant, T is the average throughput over some interval, and RTT is the average round-trip-time delay over the same interval. As the throughput increases, the utility increases, and as delays increase, the utility decreases. The choice of α determines the relative weight a user gives to throughput and delay. A delay-sensitive user will choose $\alpha \rightarrow 0$, whereas a delay-insensitive user’s $\alpha \rightarrow 1$.

In practice, a utility function may depend on a threshold. For example, a user may state that he or she is indifferent to delay, as long as it is less than 0.1 seconds. Thus, if the user gets a delay of 0.05 seconds during some interval of time, and 0.06 seconds in a later period, as far as the user is concerned, there has been no loss of utility. However, if some user’s utility *does* decrease as a result of an increase in the network load, that user will perceive the network to be congested. This motivates our definition.

Definition

A network is said to be congested from the perspective of user *i* if the utility of *i* decreases due to an increase in network load.

Remarks:

1. A network can be congested from the perspective of one user, and uncongested from the perspective of another.
2. A network can be said to be strictly uncongested if no user perceives it to be congested.
3. A user’s utility may decrease due to something other than network load, but the user may not be able to tell the difference. The onus on the user is to determine the cause of the loss of utility, and to take appropriate corrective action.

This definition is better than existing definitions since it avoids the three problems raised earlier. First, we make a clear distinction between a performance index and a congestion index. It is possible for a performance metric to decrease (for example, for RTT to increase), without a change in the congestion index (for example, if $\alpha = 1$). Second, the definition makes it clear that congestion occurs from the point of view of each individual user. Finally, the point of congestion is precisely the one where the user detects a loss of utility. No further precision is necessary, since, if the users are not dissatisfied with the available service, then the network performance, no matter how poor it is in absolute terms, is satisfactory.

Our definition places congestion control in a new light. A network that controls congestion, by our definition, must be responsive to the utility function of the users, and must be able to manage its resources so that there is no loss of utility as the load increases. Thus, the network *must* be able to differentiate between conversations, and prioritize conversations depending on the stringency of their owner’s utility. A naive approach that ignores the user’s quality-of-service requirements is automatically ruled out by this definition.

1.4. Congestion control

The previous section presented a new definition of congestion; this section describes congestion control. Two styles of control, proactive and reactive control, are presented. It is shown that congestion control must happen at several different time scales.

1.4.1. Proactive and reactive control

Congestion is the loss of utility to a user due to an increase in the network load. Hence, congestion control is defined to be the set of mechanisms that prevent or reduce such a deterioration. Practically speaking, a network can be said to control congestion if it provides each user with mechanisms to specify and obtain utility from the network. For example, if some user desires low queueing delays, then the system should provide a mechanism that allows the user to achieve this objective. If the network is unable to prevent a loss of utility to a user, then it should try to limit the loss to the extent possible, and, further, it should try to be fair to all the affected parties. Thus, in reservationless networks, where a loss of utility at high loads is unavoidable, we are concerned not only with the extent to which utility is lost, but also the degree to which the loss of utility is fairly distributed to the affected users.

A network can provide utility in one of two ways. First, it can request that each user specify a performance requirement, and can reserve resources so that this level of performance is always available to the user. This is proactive or reservation-oriented congestion control. Alternatively, users can be allowed to send data without reserving resources, but with the possibility that, if the network is heavily loaded, they may receive low utility from the network. The second method is applicable in reservationless networks. In this case, users must adapt to changes in the network state, and congestion control refers to ways in which a network can allow users to detect changes in network state, and corresponding mechanisms that adapt the user's flow to changes in this state.

In a strict proactive scheme, the congestion control mechanism is to make reservations of network resources so that resource availability is deterministically guaranteed to admitted conversations. In a reactive scheme, the owners of conversations need to monitor and react to changes in network state to avert congestion. Both styles of control have their advantages and disadvantages. With proactive control, users can be guaranteed that they will never experience loss of utility. On the other hand, to be able to make this guarantee, the number of users has to be restricted, and this could lead to underutilization of the network. Reactive control allows much more flexibility in the allocation of resources. Since users are typically not guaranteed a level of utility by the network, resources can be statistically multiplexed. However, there is always a chance that correlated traffic bursts will overload the network, causing performance degradation, and hence, congestion.

It is important to realize that proactive and reactive control are not mutually exclusive. Hybrid schemes can combine aspects of both approaches. One such hybrid scheme is for the network to provide statistical guarantees [37, 48]. For example, a user could be guaranteed an end to end delay of less than 10 seconds with 0.9 probability. Such statistical guarantees allow a network administrator to overbook resources in a controlled manner. Thus, statistical multiplexing gains are achieved, but without completely giving up performance guarantees.

Another hybrid scheme is for the network to support two types of users: guaranteed service users and best-effort users [35, 68]. Guaranteed service (GS) users are given a guarantee of quality of service, and resources are reserved for them. Best-effort (BE) users are not given guarantees and they use up whatever resources are left unutilized by GS users.

Finally, a server may reserve some minimum amount of resources for each user. Since every user has some reservation, some minimum utility is guaranteed. At times of heavy load, users compete for resources kept in a common pool [62]. Assuming some degree of independence of traffic, statistical multiplexing can be achieved without the possibility of a complete loss of utility.

1.4.2. Time scales of control

Congestion is a high-load phenomenon. The key to congestion control lies in determining the time scale over which the network is overloaded, and taking control actions on that time scale. This is explained below.

Consider the average load on a single point-to-point link. Note that the 'average load' is an *interval-based metric*. In other words, it is meaningless without also specifying the time interval over which the average is measured. If the average load is high over a small averaging interval, then the congestion control mechanism (for example, the reservation mechanism) has to deal with resource scheduling over the same small time scale. If the average load is high over a longer time scale, the congestion control mechanism needs to deal with the situation over the longer time scale *as well as* on shorter time scales.

An example should clarify this point. Consider a conversation on a unit capacity link. If the conversation is bursty, then it could generate a high load over, say, a 1ms time scale, though the average load over a 1 hour time scale could be much smaller than 1. In this case, if the conversation is delay-sensitive, then the congestion control scheme must take steps to satisfy the user delay requirement on the 1ms time scale. Over longer time scales, since the average demand is small, there is no need for congestion control.

On the other hand, if the conversation has a high average demand on the 1 hour time scale as well as the 1ms time scale, then congestion control has to be active on both time scales. For example, it may do admission control (which works over the 1 hour time scale) to make sure that network resources are available for the conversation. Simultaneously, it may also make scheduling decisions (which work on the 1ms time scale) to meet the delay requirements.

This example illustrates three points. First, congestion control must act on several different time scales simultaneously. Second, the mechanisms at each level must cooperate with each other. Scheduling policies without admission control will not ensure delay guarantees. At the same time, the admission control policy must be aware of the nature of the scheduling policy to decide whether or not to admit a conversation into the network. Third, the time scale is the time period over which a user sees changes in the network state. A congestion control mechanism that is sensitive to network state must operate on the same time scale.

We now discuss five times scales of control: those of months, one day, one session, multiple round trip times (RTTs), and less than one RTT. We believe that the design of congestion control mechanisms for each time scale should be based on sound theoretical arguments. This has the obvious advantages over an *ad hoc* approach: general applicability, ease of understanding, and formal provability of correctness. At each time scale of control, a different theoretical basis is most appropriate, and this is discussed below.

1.4.2.1. Months

Some changes to networks happen over a period of months: for example, an increase in the number of connected sites, or additional communication loads due to a new collaboration between geographically distributed sites. If these changes cause trunk lines to be substantially overloaded over long periods of time, then the only way to provide acceptable service may be to increase trunk bandwidths. Otherwise, no matter how clever the schemes at faster time scales, the network will not be able to accommodate the additional demand.

The typical response to long term overload is to lay additional bandwidth: the gradual replacement of 9600 baud serial lines with 10 Mbps Ethernets and 56 kbps trunks, and now with 100 Mbps FDDI rings and T1 trunks, is a graphic illustration of this process. Over this time scale, the formal problem corresponding to congestion control is that of capacity planning. This problem has been studied in the operations research literature. Basically, a traffic matrix, representing the volume of traffic between all pairs of sites, is constructed. A cost function describing the cost of trunk bandwidth and the utility from the network is then optimized using standard linear programming techniques to obtain an optimal capacity allocation along each path.

The problem with this approach lies in the determination of the traffic matrix. It is hard to collect all the required information, and, besides, the matrix can be quite large. Furthermore, there is no guarantee that the system workload during the next time period would be similar, so the computed solution may solve the wrong problem. Nevertheless, the use of a formal capacity planning approach would be an advance over the current *ad hoc* approach.

1.4.2.2. One day

Traffic measurements have shown that network usage exhibits cyclical behavior, with the time period of a day [13, 105]. Networks are typically lightly loaded or idle at night, but are busy from 9 am to 5 pm, reflecting the work day. A congestion control scheme that spreads the load more uniformly throughout the day will ensure that the workload is less bursty, leading to better utilization of capacity. This can be done by introducing a pricing scheme for data transfer [20, 155], and charging more for data transmitted during peak hours than during off-peak hours [7, 111, 130]. Peak-load pricing schemes used by electric and telephone utilities have much the same purpose. We believe that using ideas from economics to shape the workload on the order of a day (or several hours) is a useful form of congestion management at this time scale.

While some economic approaches have been proposed in the recent past [32, 146], we feel that the area needs much more study. Current approaches have numerous problems, such as:

- a user’s assumed utility function, or, in some approaches, demand curve, is overly simple
- the system takes a long time to reach equilibrium. For example, the SPAWN system [146] postulates multiple rounds of bidding each time a new user enters the system, and each round could take many seconds. Since new users could join every few seconds, it is not clear that the system can ever reach equilibrium.
- the approaches assume that the number of users is fixedl; in fact, given that the systems are slow to reach equilibrium, this number is like to have changed by the time equilibrium is reached
- all the users are assumed to be rational; in reality, some users may make unsophisticated, and hence irrational, decisions
- a single administrative authority is assumed
- it is time-consuming for users to bid for every resource they need
- bursty users, whose peak and average resource requirements differ considerably, are not considered.

However, the economic approach is promising, since it gives deep insights into network pricing and usage accounting that are otherwise unavailable.

1.4.2.3. Session

In connection-oriented networks, a session is the period of time between a call set-up and a call teardown. Admission control in connection oriented-networks is essentially congestion control on the time scale of a session: if the admission of a new conversation could degrade the quality of service of other conversations in the network, then the new conversation should not be admitted. This is yet another form of congestion control.

At the time a conversation is admitted to the network, the network should ensure that the resources requested by the conversation are what it really needs. Thus, the admission control scheme should give incentives to users to declare their resource needs accurately [20]. This can be designed using a variant of the standard Clark-Groves direct truth revelation mechanism (since naive users in a system with partial information would choose a dominant strategy equilibrium over more sophisticated Bayesian or sequential equilibria) [80]. The theoretical basis of this work is the area of game-theory known as ‘mechanism design’. An admission control system based on such principles has been extensively studied by Sanders [121-123].

The mechanism design approach has all the problems of the economic approach, and more. Mechanism design makes strong assumptions about the information structure in the system: for example, each switch is assumed to know the form of the utility function of each user. Further, the dynamics of the system over time are ignored. However, in the same way as economics, it can provide insights into the design of admission control.

The choice of a route at session establishment time can also be used to control congestion. If the routing establishment algorithm keeps track of network utilization along the links, it can route a new circuit along lightly loaded paths. Some proposals have exploited this idea [29, 77, 92]. Integrating feedback flow control with adaptive routing is complex, and the dynamics of their interaction are not well known. Some simplified models for the problem have been studied [29], but the applicability of these studies to real networks is unclear. In this thesis, we assume static routing, so this form of congestion control is specifically ignored.

1.4.2.4. Multiple round trip times

One round trip time (RTT) is the fundamental time constant for feedback flow control. It is the minimum time that is needed for a source of data to determine the effect of its sending rate on the network [117]. Congestion control schemes that probe network state and do some kind of filtering on the probes operate on this time scale. Examples are various window adjustment schemes [54, 63, 96, 117].

The theoretical bases for these approaches lie in queueing theory and control theory. The queueing theory approach is well studied [44], but requires strong assumptions about the network such as: Poisson arrivals from all sources, exponential service time distribution at all servers, and independence of traffic. Since observations of real networks have shown that none of these assumptions are satisfied in practice [51, 52, 104], the results obtained from a stochastic queueing approach are not entirely convincing. We believe that a naive deterministic queueing approach has several benefits, though, and this is discussed in Chapter 4.

The control-theoretic approach has also been studied in the literature, though not as thoroughly [1, 38, 66, 79, 133, 137, 138]. The drawback with existing studies is that either they are informal, and thus do not provide any formal proofs [63, 117], or they make strong assumptions about traffic behavior, service rates and so on, that do not hold in practice [126, 137, 138]. Chapter 5 presents an alternative approach that overcomes some of these problems.

Assuming that each packet is acknowledged, multiple acknowledgements can be received each RTT. If information about the state of the network is extracted from each acknowledgement, then the congestion control scheme can react to changes even faster than once per RTT. Such a scheme is described in Chapter 5, and is also made possible by a control theoretic modeling of the network.

Previous work in the area of congestion control schemes that work at the multiple RTT time scale is examined in greater detail in section 1.7.

1.4.2.5. Less than one RTT

On a scale of less than once per RTT, congestion control can be considered to be identical to scheduling data at the output queues of switches. The goal of a scheduling policy is to decide which data unit is the next to be delivered on a trunk. This choice determines the bandwidth, delay, and jitter received by each conversation, and hence the choice of the scheduling discipline is critical. A scheduling discipline that does *not* vary its allocations as a function of the network load is hence a congestion control mechanism. Examples are the Virtual Clock scheme [154], Delay-EDD [37] and Stop-and-Go queueing [47].

If the trunk bandwidth is considered as a resource, then the server implementing the scheduling discipline is essentially a resource manager that allocates bandwidths, delays and delay-jitters to each conversation. Then, it is in the best interest of a conversation to send data in such a way that it obtains the best possible utility from the server. Given that all the conversations have this objective, and the gain of one could be the loss of the other, it is clear that this framework can be cast in the form of a non-cooperative game [125]. Then, each conversation must choose a strategy (in this case, a sending rate) such that its utility is maximized. Such a model has been considered by Bharath-Kumar and Jaffe [6], Sanders [121-123], Douligeris and Majumdar [25-27] and Lazar *et al* [59]. Unfortunately, the game theoretic formulation of the problem requires utilities to be defined for each player, as well as strong information and rationality assumptions, as in the economic approach. Thus, we do not feel that it is viable in practice.

1.4.3. Need for congestion control in future networks

Congestion is a severe problem in current reservationless networks. However, in future networks the available bandwidths and switching speeds will be several orders of magnitude larger [84]. Why should congestion arise in such networks? There are several reasons:

- **Large bandwidth-delay product** : The service rate of a circuit multiplied by the round trip time determines the amount of data that a conversation must have outstanding in order to fully utilize the network. The round trip time is bounded from below by the speed-of-light propagation delay through the network. Hence, as the raw trunk bandwidth and the service rate of a conversation increases, so does the amount of outstanding data per conversation. For example, a conversation that is served at 6 Mbps and has a round trip time delay of 60 ms can have as much as 45 kbytes of data outstanding. If the service rate suddenly drops to half of its previous value, or 3 Mbps, due to cross traffic, then it takes 60 ms to react to this change, and 45/2 = 22.5 Mbytes of data can accumulate at the slowest server along the path. Since this far exceeds most buffer sizes, data loss is likely, leading to a loss of utility. Thus, a large bandwidth-delay product causes problems for reactive control and can lead to congestion.
- **Speed Mismatch** : If a switch connects a high speed line to a slower line, then a bursty conversation can, when sending data at the peak rate, fill up its buffer share, and subsequently lose packets at the switch. This creates congestion for loss-sensitive conversations. This source of congestion will persist in high-speed networks, in fact, it is probably more likely in such networks.
- **Topology** : If several input lines simultaneously send data through a switch to a single outgoing line, the outgoing line can be overloaded, leading to large queueing delays, and possible congestion for delay-sensitive traffic. This is a special case of the speed mismatch problem noted earlier.
- **Increased Usage** : Memory sizes have increased exponentially during the last decade. Yet, the demand for memory has remained, since larger memory sizes have made it feasible to develop applications that require them. Drawing a parallel to this trend, we postulate that as bandwidth increases, new applications (such as real time video) will demand these enormous bandwidths. As the available bandwidth gets saturated, the network will be operated in the high-load zone, and congestive problems are likely to reappear.
- **Misbehavior** : Congestion can be induced by misbehaving sources (such as broken sources that send a stream of back-to-back packets). Future networks must protect themselves and other sources from such misbehavior, which will continue to exist.
- **Dynamics** : As network speeds increase, the dynamics of the network also changes. Since queues can build up faster, congestive phenomena can be expected to occur much more rapidly, and perhaps have catastrophic effects [101, 132].

From these observations, we conclude that even though future networks will have larger trunk bandwidths and faster switches, congestion will not disappear.

1.5. Fundamental requirements of a congestion control scheme

We would like a congestion control scheme to have a number of properties. These are:

- Efficiency
- Ability to deal with heterogeneity
- Ability to deal with ill-behaved sources
- Stability
- Scalability
- Simplicity
- Fairness

We discuss these in turn. In the discussion, we will use the term ‘the control scheme’ to mean an integrated control scheme that simultaneously operates over all the time scales of control.

1.5.1. Efficiency

There are two aspects to efficiency. First, how much overhead does the congestion control scheme impose upon the network? As an extreme example, control packets such as the Source Quench packets in TCP/IP [112] themselves overload the congested network. We would like a control scheme to impose a minimal additional burden upon the network. This is not necessarily a binding condition: with the rapid increase in communication bandwidth, the extra load may not significantly affect network efficiency.

Second, does the control scheme lead to underutilization of critical resources in the network? Inefficient control schemes may throttle down sources even when there is no danger of congestion, leading to underutilization of resources. We would not like to operate the network suboptimally. (This raises the issue of determining what is meant by optimal utilization, but we defer the discussion to Chapter 4.)

1.5.2. Heterogeneity

As networks increase in scale and coverage, they span a range of hardware and software architectures. A control scheme that assumes a single packet size, a single transport layer protocol, or a single type of service cannot be successful in such an environment. Thus, we want the control scheme to be implementable on a wide range of network architectures.

1.5.3. Ability to deal with misbehaving sources

Note that in current networks, a network administrator does not have administrative control over the sources of messages. Thus, sources (such as users of personal workstations) are free to manipulate the network protocols to maximize the utility that they get from the network. When a switch informs a source that it should reduce its sending rate, a well-behaved source should do so. However, it is possible that an ill-behaved source may choose to ignore these signals, in the hope that this may enable it to send more data. A congestion control scheme should not fail in the presence of such hosts. In other words, badly behaving sources should not adversely affect the performance of well-behaved sources. This may require punishment of a badly behaved source, or threats of punishment that will give all sources an incentive to behave well.

1.5.4. Stability

Congestion control can be viewed as a classic negative-feedback control problem. One added complexity is that the control signals are delayed. That is, there is a finite delay between the detection of congestion and the receipt of a signal by a source. Further, the system is noisy, since observations of the system’s parameters may be corrupted by transients. These complexities may introduce instabilities into the network. Thus, we would like the control scheme to be robust, and if possible, provably stable.

1.5.5. Scalability

It is the nature of a distributed system to grow with time, and we have seen an explosive growth in network sizes in the last decade. The key to success in such an environment is scalability. We would like a congestion control scheme to scale along two orthogonal axes: bandwidth and network size.

The development of high bandwidth fiber-optic media has made it possible to build networks that have three orders of magnitude more bandwidth than networks of the recent past (45000 kbps to 56 kbps) [69]. Further, hundreds of thousands of local area networks have been interconnected in the last five years into an enormous internetwork that spans the globe. These developments make it imperative that any proposed congestion control scheme scale well on both axes.

1.5.6. Simplicity

Simplicity (unlike stupidity) is always an asset. Simple protocols are easier to implement, perhaps in hardware, and can handle increases in bandwidth. Also, simple protocols are more likely to be accepted as international standards. Thus, we would like an ideal congestion control mechanism to be simple to implement.

1.5.7. Fairness

It may be necessary for some sources to reduce their network load to control congestion. The choice of which source to throttle (either by requesting it to do so, or by dropping its packets) determines how fairly the network allocates its resources [23]. For example, if an excessive demand by some source A causes the throttling of another source B, then clearly the network is treating B unfairly. We do not want a congestion control scheme that results in unfair network resource allocation.

There are two tricky aspects of fairness: definition and implementation. The problem of defining fairness has worried network designers and welfare economists alike. Numerous fairness criteria have been proposed in the literature [43,55,106,107,115,139,149]. Each criterion has some problems, and there seems to be no absolute guide to deciding which is the best criterion to adopt [44]. Nevertheless, it is necessary to pick some justifiable criterion, since this is much better than none at all.

The second aspect is that of implementation. Implementing a fairness criterion brings in issues that lie beyond what is traditionally considered to be the scope of congestion control. For example, we may decide that it is fair to give preference to sources that are willing to pay for it. If this criterion is to be implemented, a switch needs to determine a pricing schedule and incorporate a contract negotiation and enforcement mechanism [20,155]. This leads naturally to issues in contract theory, mechanism design and incentive compatibility. Such issues have been ignored by existing congestion control schemes.

To summarize, there are a number of issues that are affected by the choice of a congestion control scheme. In this thesis, we present the design of a set of congestion control mechanisms that substantially meet the requirements posed in this section.

1.6. Fundamental assumptions

Underlying any congestion control scheme are some implicit assumptions about the network environment. These unstated assumptions largely determine the nature of the control scheme and its performance limits. We consider some of these assumptions in this section.

1.6.1. Administrative control

Can we, as designers of congestion control mechanisms, assume administrative control over the behavior of sources (for example, by dictating that only one version of a transport protocol is to be used in the network)? Or, can we assume administrative control only over the behavior of switches? Some schemes assume that we can control sources but not switches e.g. [63]. Others assume that we can control both the sources and the switches [117]. Still others assume complete control over the switches, and the ability to monitor source traffic, but no control over source traffic [54].

This assumption should be constrained by reality. In our work, we assume that we have administrative control over switches. However, source behavior is assumed to be outside our direct control (though it can be monitored, if needed). The implication is that the network must take steps to protect itself and others from malicious or misbehaving users. (Of course, users that abuse the network because of a hardware failure, such as a jammed ethernet controller, are always a threat, even when the sources can be controlled administratively.)

1.6.2. Source complexity

How complex should one assume sources to be? Since we do not have administrative control over sources, we assume that sources will perform actions that will maximize their own utility from the network. If the congestion control scheme allows intelligent users to manipulate the scheme for their own benefit, they will do so. On the other hand, users may not have the capability to respond to complex directives from the network, so we cannot assume that all users will act intelligently. In other words, while a congestion control scheme should not assume sophisticated behavior on the part of the users, at the same time, it should not be open to attack from such users.

1.6.3. Gateway complexity

Some authors assume that switches can be made complex enough to set bits on packet headers, or even determine user utility functions, whereas others assume that switches simply route packets. Ignoring monetary considerations, since we have administrative control, we can make switch control algorithms as complex as we wish, constrained only by speed requirements. It has been claimed that for high speed operation, switches should be dumb and fast. We believe that speed does not preclude complexity. What we need is a switch that is fast *and* intelligent. This can be achieved by

- having hardware support for rapid switching [100]
- optimizing for the average case [18]
- removing signaling information from the data path [41]
- choice of scheduling algorithm [156]
- an efficient call processing architecture [131].

Thus, we will assume that a switch can make fairly intelligent decisions, provided that this can be done at high speeds.

1.6.4. Bargaining power

The ultimate authority in a computer network lies in the ability to drop packets (or delay them). Since this authority lies with the switches, they ultimately have all the bargaining power. In other words, they can always coerce sources to do what they want them to do (unless this is so ridiculous that a source would rather not send any data). Any scheme that overlooks this fact loses a useful mechanism to control source behavior. Thus, schemes that treat switches and sources as peer entities (for example, [117]) are fundamentally flawed: they need to posit cooperative sources precisely because they ignore the authority that is automatically vested in switches.

1.6.5. Responsibility for congestion control

Either the sources or the switches could be made responsible for congestion control. If sources are responsible, they must detect congestion and avert it. If switches are responsible, they must take steps to ensure that sources reduce their traffic when congestion occurs, or allocate resources to avert congestion.

We believe that congestion control is a network function. If we leave the responsibility for it to sources that are not under our administrative control, then we are endangering the network. Further, the congestion detection and management functionality has to be duplicated at each of the (many) sources. In contrast, it is natural to make the fewer, controllable switches responsible for congestion control.

Note that responsibility is not the same as functionality. In other words, having responsibility for congestion does not mean that the switches have to actually perform all the actions necessary for congestion control. Switches can enforce rules that make it incentive compatible for sources to help in containing congestion. For example, a Fair Queueing [23] switch has the responsibility for congestion control, but it does congestion control by forcing *sources* to behave correctly during congestion (this is discussed in Chapter 2).

1.6.6. Bandwidth-delay product

The physical characteristics of the network play a crucial role in determining how effective a scheme will be. A congestion control scheme has to operate in some bandwidth and delay regime. When the bandwidth times round-trip-time product is small in comparison to the window size, feedback from switches to sources is feasible. However, if this product is large, buffering full windows for all users may not be feasible.

Many current schemes operate on the scale of multiple round-trip-time delays (for example, taking flow control decisions only once every two round-trip times [66]). They become infeasible when the bandwidth-delay product is large. This large product is precisely why congestion control schemes for high speed networks are so difficult, and so necessary. In this thesis, we assume that the network operates in the large bandwidth-delay product regime.

1.6.7. Traffic model

The choice of a traffic model influences the design of a congestion control scheme, since the scheme is evaluated with respect to this model. There are hidden dangers here: for example, schemes that assume Poisson sources may not robust, if, in practice, traffic does not obey this distribution. It is best to design schemes that are insensitive to the choice of the traffic model. This is achieved if a scheme does not make assumptions about the arrival distribution of packets at the switches.

What should be the traffic model? We do not have much data at our disposal, since there are no high-speed WANs yet available to measure. However, there are three trends that point to a reasonable model. First, the move towards integration of data, telephony and video services indicates that some number of sources in our environment will be phone and video sources. These can generate high bandwidth traffic over periods of time spanning minutes or hours.

Second, existing studies have shown that data traffic is very bursty [52, 104]. This tendency will certainly be exaggerated by increases in line speeds. Finally, we note that current applications are mostly of two sorts - low bandwidth interactive conversations, and high bandwidth offline bulk data transfer [12]. At higher speeds, the bulk data transfers that last several seconds today will collapse into bursts. This reinforces our belief that future traffic will basically be bursty.

To sum up, we expect that the traffic will be generated by two kinds of sources: one that demands a sustained high bandwidth, and the other that generates bursts of traffic at random intervals of time. We call these sources ‘FTP’ and ‘Telnet’ in this thesis (these terms are probably outdated, but we use them for convenience). This model is fairly simple, and does not involve any assumptions about packet arrival distributions. Thus, schemes that work for this model will probably work for a large variety of parametrically constrained models as well (e.g. for traffic where the bursts are exponentially or uniformly distributed). Since the traffic characteristics for future networks are still unknown, this model is speculative. However, we think that it is as reasonable as any that have so far been studied.

Summary of our assumptions

- We assume that we have administrative control over switches, but not over sources.
- Some sources will be intelligent enough to exploit any flaws in the design, but not all the sources will be able to implement complex congestion and flow control strategies.
- Switches should be fast, but they should carry out intelligently designed congestion control strategies.
- The switches should not be treated as peer entities of sources, their inherent capacity to delay or drop packets should be used to coerce sources to behave in a socially acceptable manner.
- The responsibility for congestion control should lie with the switches, sources cannot be trusted with this job.
- The scheme should operate in a regime where the delay bandwidth product is large as compared to current window sizes.
- Traffic consists of two types of sources, ones that generate sustained high bandwidth traffic, and others that generate intermittent bursts.

1.7. Previous work

This section surveys previous work in congestion control at the session, multiple RTT and less than one RTT time scales. Research in these areas of congestion control has mushroomed in the recent past. Consequently, this survey cannot claim to be complete. We have tried to mention all the research efforts that we are aware of, but it is almost certain that some others have been overlooked.

While our work stresses the need for a network to be sensitive to a user’s utility, previous work on congestion control has concentrated mainly on mechanisms for avoiding packet losses and reducing queueing delay. The efforts have been strongly oriented towards either reservationless networks or reservation-oriented networks. For reservation-oriented networks, the work had been at the session and scheduling time scales, and for reservationless networks exclusively at the multiple RTT time scale. We now discuss previous work for each class of network.

1.7.1. Reservationless networks

In reservationless networks, control has to be reactive. A reactive congestion control scheme is implemented at two locations: at the switches, where congestion occurs, and at the sources, which control the net inflow of packets into the network. Typically, a switch uses some metric (such as overflow of buffers) to determine the onset of congestion, and implicitly or explicitly communicates this problem to the sources, which reduce their input traffic. Even in this simplified picture, a few problems are apparent.

How is congestion to be detected? (Congestion Detection)

How is the problem signaled to sources? (Communication)

What actions do the switches take? (Decongestion)

Which sources are held responsible for congestion? (Selection)

What actions must the sources take? (Flow Control)

What if some sources ignore these signals? (Enforcement)

These questions must be answered by every reactive congestion control scheme. Depending upon the choices made in answering each question, a variety of schemes have been proposed.

Comprehensive surveys by Gerla and Kleinrock [44] and Pouzin [110] discuss numerous congestion control schemes. However, the taxonomies they develop are oriented towards classifying flow control techniques, and are not appropriate in our work. Instead, we will base our survey on the questions raised earlier.

1.7.1.1. Congestion detection

How is a switch or source to detect congestion? There are several alternatives.

- The most common one is to notice that the output buffers at a switch are full, and there is no space for incoming packets. If the switch wishes to avoid packet loss, congestion avoidance steps can be taken when some fraction of the buffers are full (such as in the FQbit scheme discussed in Chapter 2). A time average of buffer occupancy can help smooth transient spikes in queue occupancy [116, 117].
- A switch may monitor output line usage. It has been found that congestion occurs when trunk usage goes over a threshold (typically 90%) and so this metric can be used as a signal of impending congestion (as in CIGALE or Cyclades) [61, 88]. The problem with this metric is that congestion avoidance could keep the output line underutilized, leading to possible inefficiency.
- A source may monitor round-trip delays. An increase in these delays signals an increase in queue sizes, and possible congestion [66].
- A source may probe the network state using some probing scheme (for example, the packet-pair method described in Chapter 4).
- A source can keep a timer that sets off an alarm when a packet is not acknowledged ‘in time’ [108]. When the alarm goes off, congestion is suspected [153].

1.7.1.2. Communication

Communication of congestion information from the congested switch to a source can be implicit or explicit. When communication is explicit, the switch sends information in packet headers [115] or in control packets such as Source Quench packets [112], choke packets [88], state-exchange packets [79, 120], rate-control messages [148], or throttle packets [70] to the source. Implicit communication occurs when a source uses probe values [75], retransmission timers [153], throughput monitoring [147], or delay monitoring [66] to indicate the (sometimes only suspected) occurrence of congestion.

Explicit communication imposes an extra burden on the network, since the network needs to transmit more packets than usual, and this may lead to a loss in efficiency. On the other hand, with implicit communication, a source may not be able to distinguish between congestion and other performance problems, such as a hardware problem [153]. Thus, the communication channel is quite noisy, and a cause of potential instability.

1.7.1.3. Switch action

An overloaded switch can signal impending congestion to the sources, and, at worst, can drop packets. In virtual circuit network layers, hop level flow control can throttle upstream virtual circuits. In any case, the problem is to select either the source to throttle, or whose packets to drop. The fairness of the congestion control scheme depends on this choice made by the switches.

Schemes that rely on line usage as a congestion metric throttle all sources sending packets on an overloaded link [14, 110]. This scheme is unfair in the sense that sources that use a small fraction of the congested link are punished as severely as large users. The Loss-load curve scheme [148] and the selective DECbit scheme [115] seek to overcome this problem by computing the share of the load due to each user, and selectively dropping packets from that source.

If buffer usage is a congestion metric, switches drop packets or throttle sources when a source exceeds its share of buffers. This share is determined by the buffer allocation strategy, and the rate at which the buffers are emptied depends upon the service discipline. Thus, the buffer allocation strategy and the service discipline jointly determine which sources are affected (with the three exceptions noted below). We now discuss these two aspects of switch behavior.

The optimal buffer allocation scheme has received a lot of attention in the literature, and is surveyed in [44]. Overall, the conclusion in that survey is that a scheme where each output line is guaranteed a minimum number of buffers, and is not allowed to exceed a maximum, is fair and efficient. Other schemes, such as Input Buffer Limit [81], Channel Queue Limit [70], Buffer Class [45] and Matsumoto’s X.75 proposal [91] discriminate against some sources, and are thus unfair.

Queue service disciplines are implemented in the servers on the output queues of packet switches [100]. The choice of service discipline influences the kind of performance guarantees that can be made to network clients in reservation-oriented networks [156]. We defer a discussion of the choice of service discipline in reservationless networks to Chapter 2.

We claimed earlier that the choice of which source or conversation is affected by congestion is implicitly determined by the service mechanism and the buffer allocation policy. Some exceptions are in the DEC scheme [115], the Loss-load curve approach [148] and the Random dropping scheme [90]. These schemes explicitly try to be fair in allocating responsibility for congestion to sources. In the DECbit and Loss-load schemes, a switch maintains state information about the demand for bandwidth by each source and its fair share. When the demand exceeds the fair share, and the buffers are getting filled, that source is penalized. This is fair by design. Random dropping is an ad hoc technique to achieve fairness. The essential idea is that, when a buffer is full, the packet to drop is selected at random. Hence, users who use more bandwidth, and will probably occupy a larger fraction of the buffer, have a higher chance of packet loss. However, work by Mankin has shown that, by itself, this scheme does not reduce congestion [90].

1.7.1.4. Flow control

A number of congestion control schemes have been proposed that operate at the sources. These schemes use the loss of a packet (or the receipt of choke information) to reduce the source sending rate in some way. The two main types of schemes are choke schemes and rate-control schemes.

In a choke scheme, a source shuts down when it detects congestion. After some time, the source is allowed to start up again [63, 88]. Choking is not efficient, since the reaction of the sources is too abrupt. The stability of the choke scheme has not been analyzed, but the simple example given in §1.5.4 seems to indicate that the scheme is prone to oscillations.

In a rate-control scheme, when a source detects congestion it reduces the rate at which it sends out packets, either using a window adjustment scheme [117] or a rate adjustment scheme [17, 75]. The latter is particularly suitable for sources that do rate based flow control. The advantage of rate control schemes over choke schemes is that rate control allows a gradual transition between sending no packets at all to sending full blast. Rate control seems to be an attractive alternative for congestion control. Detailed analysis of the stability and efficiency of rate control are presented in Chapters 5 and 6.

1.7.1.5. Enforcement

Congestion is a social problem and, unless the performance perceived by each source is decoupled from the performance perceived by other sources, every source must cooperate in solving it. However, in the large non-trustable, non-cooperative networks of the future, such cooperation can no longer be assumed [118]. Thus, solutions that are predicated upon cooperative sources are not satisfactory.

Many current schemes do not study enforcement. For example, in the DECbit scheme [117], if a source chooses to ignore the congestion avoidance bits set by a switch, then the other cooperative sources will automatically give up bandwidth to the ill-behaved source. The same is true for the Jacobson-Karels TCP scheme [63]. This problem is avoided partially by the Random-drop scheme [90], and completely by the Loss-load curve method [148].

There is a need for solutions that will work in the presence of ill-behaved hosts. That is, a congestion control decision must not only be communicated to the sources, it must also be enforced. Of all existing proposals, only the ones based on the round-robin (such as Fair Queueing, or Earliest Due Date) service discipline have this property. This is discussed in Chapter 2.

1.7.2. Reservation-oriented networks

In reservation-oriented networks, network resources can be allocated at the start of each session. Then, the network can guarantee a performance level to a conversation by performing admission control. This can guarantee congestion control, but perhaps at the cost of underutilization of network resources.

One of the earliest schemes to prevent packet losses was developed for the Datakit network [41]. Here, the network places a limit on the size of the flow control window of each conversation, and the connection establishment packet reserves a full window’s worth of buffer space at each intermediate switch. Thus, every virtual circuit, once established, is guaranteed to find enough buffers for each outstanding packet, and packet loss is avoided.

The complementary scheme is to reserve bandwidth instead of buffers. This is the approach taken by Zhang in the Flow Network [154], by Ferrari *et al* in their real-time channel establishment scheme [37], by Topolcic *et al* in the ST-II proposal [134] and by Cidon *et al* for the PARIS network [16]. In a hybrid scheme described in reference [24] a source makes a reservation for buffers at the beginning of a call, and a reservation for bandwidth before the start of each burst. This allows bandwidth to be efficiently shared, but each burst experiences a round trip time delay.

There are four major problems with any naive reservation scheme: scaling, queueing delay, underutilization and enforcement.

1.7.2.1. Scaling

For large and high speed networks, each switch needs a considerable amount of memory. For example, if the RTT is 70ms over a 1Gbps fiber, one may need as much as 70Mbits of buffering per conversation per switch! The two-window scheme proposed by Hahne *et al* [54] greatly reduces this memory demand, and makes buffer reservation quite attractive. Work by Mitra *et al* has shown that, theoretically, this requirement can be reduced even further [96]. Another way to reduce buffer requirements is to require the user to obey some traffic profile [34].

1.7.2.2. Delay

In a proactive scheme, at overload, a full window could be buffered at the bottleneck. When this happens, the queueing delay could be unacceptable. Delays can be bounded by computing the worst-case delay at the time of call set-up, and doing admission control [37].

1.7.2.3. Underutilization

The major problem with reservations is that the network could be underutilized: an overzealous admission control scheme could prevent congestion by allowing only a few conversations to enter. This is not acceptable. The crux of the problem lies in determining how many conversations can be admitted into the network without reducing the performance guarantees made to the existing conversations. This has been studied by Ferrari et al [37]. One solution to the problem is to define statistical guarantees, where some degree of performance loss can be tolerated [35]. Conversations with statistical guarantees can be statistically multiplexed. Other hybrid schemes have been described in section 1.4.1.

1.7.2.4. Enforcement

How can we ensure that conversations actually send data at their reserved rate? There are two types of enforcement mechanisms: those that act at the network access point, and those that act at each switch.

Some researchers, especially from the telecom community, have postulated the existence of Network Access Points (NAPs) where users (subscribers) can access a homogeneous switching fabric. The NAPs can monitor and shape user traffic. If some users violate their stated traffic envelope, their data can be dropped or violation-tagged [5]. An efficient way to monitor, and, if necessary, reshape user traffic behavior to make it less bursty, is the leaky-bucket scheme [87, 128, 136, 141].

Other schemes do enforcement at each switch. This is through some form of round-robin-like queueing discipline at each switch. In the Flow network, the Virtual Clock mechanism is used [154], and in the real-time channel establishment proposal, the discipline is Earliest-Due-Date with distributed flow control [37]. Ferrari and Verma [36] and Kanakia [68] have proposed mechanisms based on calendar queues [11] that have a similar effect.

1.7.3. Quality of service

One can view congestion control as being able to guarantee quality of service at high loads. There has been some previous work in guaranteeing quality of service in networks.

Postel made an early suggestion for reservationless networks, though this was not studied in any depth [113]. Golestani's Stop-and-go queueing provides conversations with bandwidth, delay and jitter bounds, and is similar in spirit to Kalmanek *et al*'s Hierarchical Round Robin scheme [46, 47, 68]. Ferrari *et al* have proposed schemes that deliver deterministic and statistical guarantees for delay, bandwidth, packet loss and jitter [37, 142]. Other related work is that of Lazar, who has proposed a network that offers classes of service, where each class has a performance guarantee [83]. Note that having only a small number of classes of service, instead of letting each user define its own performance requirements, makes implementation easier, but does not adequately satisfy our definition of congestion control. Work by Estrin and Clark to ensure that a conversation can determine a route that best satisfies its quality of service requirement (policy routing) is also relevant [19, 31].

1.8. Scope of the thesis

As our survey indicates, the study of congestion control is a large and rapidly expanding area. Since it is impossible to study the entire area in any depth in a single dissertation, we limit our scope to congestion control in reservationless networks and only at the multiple-RTT and faster than one RTT time scales. The techniques developed in this work can be used for the 'best-effort' data traffic in reservation-oriented networks.

We initially turn our attention to the problem of protecting well-behaved users from ill-behaved ones by allocating all users a fair share of the network bandwidth (Chapter 2). We show that this Fair Queueing discipline not only ensures that each user gets a fair share of the network bandwidth, but also enables users to probe the network state. After describing techniques for the efficient implementation of Fair Queueing (Chapter 3), we present a novel state probing technique that a source can use in networks of Fair Queueing servers (Chapter 4). A flow control algorithm based on control theoretic principles that uses the probe values to do predictive control is then described in Chapter 5. Chapter 6 provides simulation studies to back up our claims, and Chapter 7 presents our conclusions.

Our original contributions are in the theoretical modeling of the congestion and congestion control problems, deriving theoretically sound solutions, and using these to develop practical algorithms. These algorithms are tested using a network simulator [73] on a set of eight test scenarios.

Chapter 2: Fair Queueing

2.1. Introduction

Datagram networks have long suffered from performance degradation in the presence of congestion [44]. The rapid growth, in both use and size, of computer networks has sparked a renewed interest in methods of congestion control. These methods have two points of implementation. The first is at the source, where flow control algorithms vary the rate at which the source sends packets. Flow control algorithms are designed primarily to ensure the presence of free buffers at the destination host, but we are more concerned with their role in limiting the overall network traffic, and in providing users with maximal utility from the network.

The second point of implementation is at the switches. Congestion can be controlled at the switches through routing and queueing algorithms. Adaptive routing, if properly implemented, lessens congestion by routing packets away from network bottlenecks. Queueing algorithms, which control the order in which packets are sent and the usage of the switch's buffer space, determine the way in which packets from different sources interact with each other. This, in turn, affects the collective behavior of flow control algorithms. We argue that this effect, which is often ignored, makes queueing algorithms a crucial component in effective congestion control.

Queueing algorithms can be thought of as allocating three nearly independent quantities: bandwidth (*which* packets get *transmitted*), promptness (*when* do those packets get *transmitted*), and buffer space (*which* and *when* packets get *discarded* by the switch). Currently, the most common queueing algorithm is first-come-first-serve (FCFS). In this scheme, the order of arrival completely determines the bandwidth, promptness, and buffer space allocations, inextricably intertwining these three allocation issues. Since each user may have a different preference for each allocated quantity, FCFS queueing cannot provide adequate congestion management.

There may be flow control algorithms that can, when universally implemented in a network with FCFS switches, overcome these limitations and provide reasonably fair and efficient congestion control. However, with today's diverse and decentralized computing environments, it is unrealistic to expect universal implementation of any given flow control algorithm. This is not merely a question of standards, but also one of compliance. First, even if a universal standard such as ISO [12] were adopted, malfunctioning hardware and software could violate the standard. Second, there is always the possibility that individuals would alter the algorithms on their own machine to improve their performance at the expense of others. Consequently, congestion control algorithms should function well even in the presence of ill-behaved sources.

Unfortunately, irrespective of the flow control algorithm used by the well-behaved sources, networks with FCFS switches do not have this property. A single source, sending packets to a switch at a sufficiently high speed, can capture an arbitrarily high fraction of the bandwidth of the outgoing line. Thus, FCFS queueing is not adequate; more discriminating queueing algorithms must be used in conjunction with source flow control algorithms to control congestion effectively in noncooperative environments.

Following a similar line of reasoning, Nagle [101] proposed a *fair queueing* (FQ) algorithm in which switches maintain separate queues for packets from each individual source. The queues are serviced in a round-robin manner. This prevents a source from arbitrarily increasing its share of the bandwidth or the queueing delay received by the other sources. In fact, when a source sends packets too quickly, it merely increases the length of its own queue. Nagle's algorithm, by changing the way packets from different sources interact, does not reward, nor leave sources vulnerable to, anti-social behavior. This proposal appears to have considerable merit, and this chapter describes a modification of Nagle's scheme and explores its implications in some depth.

The three different components of congestion control algorithms introduced above, source flow control, switch routing, and switch queueing algorithms, interact in interesting and complicated ways. It is impossible to assess the effectiveness of any algorithm without reference to the other components of congestion control in operation. We will evaluate our proposed queueing algorithm in the context of static routing and several widely used flow control algorithms. The aim is to find a queueing algorithm that functions well in current computing environments. The algorithm might, indeed it should, *enable* new and improved routing and flow control algorithms (as shown in Chapters 4 and 5), but it must not require them.

In circuit switched networks, where there are explicit buffer reservation and uniform packet sizes, it has been established that round robin service disciplines allocate bandwidth fairly [55, 72]. Recently Morgan [98] has examined the role such queueing algorithms play in controlling congestion in circuit switched networks; while his application context is quite different from ours, his conclusions are qualitatively similar. In other related work, the Datakit queueing algorithm combines round robin service and FIFO priority service, and has been analyzed extensively [42, 86]. Similar work has been presented by Zhang [154]; her *Virtual Clock* switch queueing algorithm is essentially identical to the fair queueing algorithm presented here [157]. Zhang analyzes this algorithm in the context of a proposed resource reservation scheme, the *Flow Network*, whereas we do not consider resource reservation.

2.2. Fair Queueing

2.2.1. Motivation

What are the requirements for a queueing algorithm that will allow source flow control algorithms to provide users with adequate utility even in the presence of ill-behaved sources? We start with Nagle's observation that a queueing algorithm must provide *protection*, so that ill-behaved sources can only have a limited negative impact on well-behaved sources. Allocating bandwidth and buffer space in a *fair* manner, to be defined later, automatically ensures that ill-behaved sources can get no more than their fair share. This led us to adopt, as our central design consideration, the requirement that the queueing algorithm allocate bandwidth and buffer space fairly. Ability to control the promptness, or delay, allocation somewhat independently of the bandwidth and buffer allocation is also desirable. Finally, we require that the switch should provide service that, in some sense, does not depend discontinuously on a packet's time of arrival (this continuity condition will be made precise when we define our algorithm). This continuity requirement attempts to prevent the efficiency of source flow control implementations from being overly sensitive to timing details (timers are the Bermuda Triangle of flow control algorithms).

Nagle's proposal does not satisfy these requirements. The most obvious flaw is its lack of consideration of packet lengths. A source using long packets gets more bandwidth than one using short packets, so bandwidth is not allocated fairly. Also, the proposal has no explicit promptness allocation other than that provided by the round-robin service discipline. In addition, the static round robin ordering violates the continuity requirement. These defects are corrected in our version of fair queueing, which we define after first discussing our definition of fairness.

In stating our requirements for queueing algorithms, we have left the term *fair* undefined. The term *fair* has a clear colloquial meaning, but it also has a technical definition (actually several, as discussed in Chapter 1, but only one is considered here). Consider, for example, the allocation of a single resource among N users. Assume there is an amount μ_{total} of this resource, and that each of the users requests an amount ρ_i and, under a particular allocation, receives an amount μ_i. What is a fair allocation? The max-min fairness criterion [43, 55, 115] states that an allocation is fair if (1) no user receives more than its request, (2) no other allocation scheme satisfying condition 1 has a higher minimum allocation, and (3) condition 2 remains recursively true as we remove the minimal user and reduce the total resource accordingly, i.e. $\mu_{total} \leftarrow \mu_{total} - \mu_{\min}$. This condition reduces to $\mu_i = MIN(\mu_{fair}, \rho_i)$ in the simple example, with μ_{fair}, the *fair share*, being set so that $\mu_{total} = \sum_{i=1}^{N} \mu_i$. This concept of fairness easily generalizes to the multiple resource case [115]. Note that implicit in the max-min definition of fairness is the assumption that the users have equal *rights* to the resource.

In this application, the bandwidth and buffer demands are clearly represented by the packets that arrive at the switch. (Demands for promptness are not explicitly communicated, and we return to this issue later.) However, it is not clear what constitutes a *user*. The user associated with a packet could refer to the source of the packet, the destination, the source-destination pair, or even refer to an individual process running on a source host. Each of these definitions has limitations. Allocation per source unnaturally restricts sources such as file servers, which typically consume considerable bandwidth. Ideally, the switches could know that some sources deserve more bandwidth than others, but there is no adequate mechanism for establishing that knowledge in today's networks. Allocation per receiver allows a receiver's useful incoming bandwidth to be reduced by a broken or malicious source sending unwanted packets to it. Allocation per process on a host encourages human users to start several processes communicating simultaneously, thereby evading the original intent of fair allocation. Allocation per source-destination pair allows a malicious source to consume an unlimited amount of bandwidth by sending many packets all to different destinations. While this does not allow the malicious source to do useful work, it can prevent other sources from obtaining sufficient bandwidth.

Overall, allocation on the basis of source-destination pairs, or *conversations*, seems the best tradeoff between security and efficiency and will be used here. However, our treatment will apply to any of these interpretations of the notion of user. Given the requirements for an adequate queueing algorithm, coupled with the definitions of *fairness* and *user*, we now turn to the description of our fair queueing algorithm.

2.2.2. Definition

It is simple to allocate buffer space fairly by dropping packets, when necessary, from the conversation with the largest queue. Allocating bandwidth fairly is less straightforward. Pure round-robin service provides a fair allocation of packets sent, but fails to guarantee a fair allocation of bandwidth because of variations in packet sizes. To see how this unfairness can be avoided, we first consider a hypothetical service discipline where transmission occurs in a bit-by-bit round robin (BR) fashion (as in a head-of-queue processor sharing discipline). This service discipline allocates bandwidth fairly since at every instant in time each conversation is receiving its fair share. Let $R(t)$ denote the number of rounds made in the round-robin service discipline up to time t ($R(t)$ is a continuous function, with the fractional part indicating partially completed rounds). Let $N_{ac}(t)$ denote the number of active conversations, i.e. those that have bits in their queue at time t. Then, $\frac{\partial R}{\partial t} = \frac{\mu}{N_{ac}(t)}$, where μ is the speed of the switch's outgoing line (we will, for convenience, work in units such that $\mu = 1$). A packet of size P whose first bit gets serviced at time t_0 will have its last bit serviced P rounds later, at time t such that $R(t) = R(t_0)+P$. Let t_i^α be the time that packet i belonging to conversation α arrives at the switch, and define the numbers S_i^α and F_i^α as the values of $R(t)$ when the packet started and finished service. With P_i^α denoting the size of the packet, the following relations hold: $F_i^\alpha = S_i^\alpha + P_i^\alpha$ and $S_i^\alpha = MAX(F_{i-1}^\alpha, R(t_i^\alpha))$. Since $R(t)$ is a strictly monotonically increasing function whenever there are bits waiting to be sent at the switch, the ordering of the F_i^α values is the same as the ordering of the finishing times of the various packets in the BR discipline.

Sending packets in a bit-by-bit round robin fashion, while satisfying our requirements for an adequate queueing algorithm, is obviously unrealistic. We hope to emulate this impractical algorithm by a practical packet-by-packet transmission scheme. Note that the functions $R(t)$ and $N_{ac}(t)$ and the quantities S_i^α and F_i^α depend only on the packet arrival times t_i^α and not on the actual packet transmission times, as long as we define a conversation to be active whenever $R(t) \le F_i^\alpha$ for $i = MAX(j \mid t_j^\alpha \le t)$. We are thus free to use these quantities in defining our packet-by-packet transmission algorithm. A natural way to emulate the bit-by-bit round-robin algorithm is to let the quantities F_i^α define the sending order of the packets. Our packet-by-packet transmission algorithm is simply defined by the rule that, whenever a packet finishes transmission, the next packet sent is the one with the smallest value of F_i^α. The continuity requirement mentioned earlier can be restated precisely as demanding that the relative transmission priorities depend continuously on the packet arrival times. The fact that the F_i^α's depend continuously on the t_i^α's means that our algorithm satisfies this continuity requirement.

In a preemptive version of this algorithm, newly arriving packets whose finishing number F_i^α is smaller than that of the packet currently in transmission preempt the transmitting packet. For practical reasons, we have implemented the nonpreemptive version, but the preemptive algorithm (with resumptive service) is more tractable analytically. Clearly the preemptive and nonpreemptive packetized algorithms do not give the same instantaneous bandwidth allocation as the BR version. However, for each conversation the total bits sent at a given time by these three algorithms are always within $P_{\max}$ of each other, where $P_{\max}$ is the maximum packet size (this emulation discrepancy bound is proved in reference [50]). Thus, over sufficiently long conversations, the packetized algorithms asymptotically approach the fair bandwidth allocation of the BR scheme.

Recall that a user's request for promptness is not made explicit. The IP protocol [108] does have a field for type-of-service, but not enough applications make intelligent use of this option to render it a useful hint. Consequently, promptness allocation must be based solely on data already available at the switch. One such allocation strategy is to give more promptness (less delay) to users who utilize less than their fair share of bandwidth. Separating the promptness allocation from the bandwidth allocation can be accomplished by introducing a nonnegative parameter δ, and defining a new quantity, the *bid* B_i^α, as $B_i^\alpha = P_i^\alpha + MAX(F_{i-1}^\alpha, R(t_i^\alpha) - \delta)$. The quantities $R(t)$, $N_{ac}(t)$, F_i^α, and S_i^α remain as before, but now the sending order is determined by the B's, not the F's. The asymptotic bandwidth allocation is independent of δ, since the F's control the bandwidth allocation, but the algorithm gives slightly faster service to packets belonging to an inactive conversation. The parameter δ controls the extent of this additional promptness. Note that the bid B_i^α is continuous in t_i^α, so that the aforementioned continuity requirement is met.

The role of this term δ can be seen more clearly by considering the two extreme cases $\delta = 0$ and $\delta = \infty$. If an arriving packet has $R(t_i^\alpha) \le F_{i-1}^\alpha$, then the conversation α is active (i.e. the corresponding conversation in the BR algorithm would have bits in the queue). In this case, the value of δ is irrelevant and the bid number depends only on the finishing number of the previous packet. However, if $R(t_i^\alpha) > F_{i-1}^\alpha$, so that the α conversation is inactive, the two cases are quite different. With $\delta = 0$, the bid number is given by $B_i^\alpha = P_i^\alpha + R(t_i^\alpha)$, and is completely independent of the previous history of user α. With $\delta = \infty$, the bid number is $B_i^\alpha = P_i^\alpha + F_{i-1}^\alpha$ and depends only the previous packet's finishing number, no matter how many rounds ago. For intermediate values of δ, scheduling decisions for packets of inactive conversations depends on the previous packet's finishing round as long as it was not too long ago, and δ controls how far back this dependence goes.

Recall that when the queue is full and a new packet arrives, the last packet from the conversation currently using the most buffer space is dropped. We have chosen to leave the quantities F_i^α and S_i^α unchanged when we drop a packet. This provides a small penalty for ill-behaved hosts, in that they will be charged for throughput that, because of their own poor flow control, they could not use. Recent work [57] raises questions about the desirability of this aspect of our algorithm.

2.2.3. Performance

The desired bandwidth and buffer allocations are completely specified by the definition of fairness, and we have demonstrated that our algorithm achieves those goals. However, we have not been able to characterize the promptness allocation for an arbitrary arrival stream of packets. To obtain some quantitative results about the promptness, or delay, performance of a single FQ switch, we consider a very restricted class of arrival streams in which there are only two types of sources. There are FTP-like file transfer sources, which always have ready packets and transmit them whenever permitted by the source flow control (which, for simplicity, is taken to be sliding window flow control), and there are Telnet-like interactive sources, which produce packets intermittently according to some unspecified generation process. What are the quantities of interest? An FTP source is typically transferring a large file, so the quantity of interest is the transfer time of the file, which for asymptotically large files depends only on the bandwidth allocation. Given the configuration of sources this bandwidth allocation can be computed *a priori* by using the fairness property of FQ switches. The interesting quantity for Telnet sources is the average delay of each packet, and it is for this quantity that we now provide a rather limited result.

Consider a single FQ switch with N FTP sources sending packets of size P_F, and allow a single packet of size P_T from a Telnet source to arrive at the switch at time t. It will be assigned a bid number $B = R(t)+P_T-\delta$; thus, the dependence of the queueing delay on the quantities P_T and δ is only through the combination $P_T-\delta$. We will denote the queueing delay of this packet by $\phi(t)$, which is a periodic function with period NP_F. We are interested in the average queueing delay Δ

$$\Delta \equiv \frac{1}{NP_F} \int_0^{NP_F} \phi(t)\,dt$$

The finishing numbers F_i^{α} for the N FTP's can be expressed, after perhaps renumbering the packets, by $F_i^{\alpha} = (i+l^{\alpha})P_F$ where the l's obey $0 \le l^{\alpha} < 1$. The queueing delay of the Telnet packet depends on the configuration of l's whenever $P_T < P_F$. One can show that the delay is bounded by the extremal cases $l^{\alpha} = 0$ for all α and $l^{\alpha} = \alpha / N$ for $\alpha = 0, 1, \ldots, N-1$. The delay values for these extremal cases are straightforward to calculate; for the sake of brevity we omit the derivation and merely display the result below. The average queueing delay is given by $\Delta = A(P_T-\delta)$, where the function $A(P)$, the delay with $\delta = 0$, is defined below (with integer k and small constant ε, $0 \le \varepsilon < 1$, defined via $P_T = P_F(k+\varepsilon)/N$).

Preemptive

$$A(P) = N(P - \frac{P_F}{2}) \qquad \text{for} \quad P \ge P_F$$

$$N(P - \frac{P_F}{2}) \le A(P) \le \frac{NP^2}{2P_F} \quad \text{for} \;\; P_F \ge P \ge \frac{P_F}{2}(1 + \frac{1}{N})$$

$$\frac{1}{2P_F}(\frac{P_F}{2} + N(P - \frac{P_F}{2}))^2 \le A(P) \le \frac{NP^2}{2P_F} \quad \text{for} \;\; \frac{P_F}{2}(1 + \frac{1}{N}) \ge P \ge \frac{P_F}{2}(1 - \frac{1}{N})$$

$$0 \le A(P) \le \frac{NP^2}{2P_F} \qquad \text{for} \quad \frac{P_F}{2}(1 - \frac{1}{N}) \ge P$$

Nonpreemptive

$$A(P) = N(P - \frac{P_F}{2}) \qquad \text{for} \quad P \ge P_F$$

$$N(P - \frac{P_F}{2}) \le A(P) \le (\frac{P_F}{2})\left\{ 1 + \frac{1}{N}[k^2 + k(2\varepsilon - 1)] \right\} \;\; \text{for} \;\; P_F \ge P \ge \frac{P_F}{2}(1 + \frac{1}{N})$$

$$\frac{P_F}{2} \le A(P) \le (\frac{P_F}{2})\left\{ 1 + \frac{1}{N}[k^2 + k(2\varepsilon - 1)] \right\} \;\; \text{for} \;\; \frac{P_F}{2}(1 + \frac{1}{N}) \ge P$$

A more detailed analysis of the single server case can be found in reference [23]. What happens in a network of FQ switches? There are few analytical results here, but Hahne [55] has shown that for strict round robin service switches and only FTP sources there is fair allocation of bandwidth (in the multiple resource sense) when the window sizes are sufficiently large. She also provides examples where insufficient window sizes, but much larger than the pipeline depth of the communication path, can result in unfair allocations. It can be shown that both of these properties hold for our fair queueing scheme.

Chapters 4 and 5 analyze networks of FQ switches from the point of view of a single conversation. While this analysis cannot explain overall network dynamics, it is sufficient to model and control a single conversation. Simulation results for networks of FQ switches are presented in Chapter 6.

2.3. Discussion

In an FCFS switch, the queueing delay of packets is, on average, uniform across all sources and directly proportional to the total queue size. Thus, achieving ambitious performance goals, such as low delay for Telnet-like sources, or even mundane ones, such as avoiding dropped packets, requires coordination among all sources to control the queue size. Having to rely on source flow control algorithms to solve this control problem, which is extremely difficult even in a maximally cooperative environment and impossible in a noncooperative one, merely reflects the inability of FCFS switches to distinguish between users and to allocate bandwidth, promptness, and buffer space independently.

In the design of the fair queueing algorithm, we have attempted to address these issues. The algorithm does allocate the three quantities separately. Moreover, the promptness allocation is not uniform across users and is somewhat tunable through the parameter δ. Most importantly, fair queueing creates a firewall that protects well-behaved sources from their uncouth brethren. Not only does this allow the current generation of flow control algorithms to function more effectively, but it creates an environment where users are rewarded for devising more sophisticated and responsive algorithms. The game-theoretic issue first raised by Nagle, that one must change the rules of the switch's game so that good source behavior is encouraged, is crucial in the design of switch algorithms [101]. A formal game-theoretic analysis of a simple switch model (an exponential server with N Poisson sources) suggests that fair queueing algorithms make self-optimizing source behavior result in fair, protective, nonmanipulable, and stable networks; in fact, they may be the only reasonable queueing algorithms to do so [124]. Our calculations show that the fair queueing algorithm is able to deliver low delay to sources using less than their fair share of bandwidth, and that this delay is insensitive to the window sizes being used by the FTP sources.

The protection property of the FQ algorithm enables a new class of flow control algorithm. Since each user gets a fair share of the network bandwidth, the stream of acknowledgments received by the source, is, to a first approximation, dependent only on that source's own behavior. Hence, by monitoring the acknowledgment stream, the source can optimize its sending rate regardless of the behavior of the other sources. This idea is explained in greater detail in Chapters 4 and 5.

In this chapter we have compared our fair queueing algorithm with only the standard first-come-first-serve queueing algorithm. We know of three other widely known queueing algorithm proposals. The first two were not intended as a general purpose congestion control algorithms. Prue and Postel [113] have proposed a type-of-service priority queueing algorithm, but allocation is not made on a user-by-user basis, so fairness issues are not addressed. There is also the Fuzzball selective preemption algorithm [94, 95] whereby the switches allocate buffers fairly (on a source basis, over all of the switch's outgoing buffers). This is very similar to our buffer allocation policy, and so can be considered a subset of our FQ algorithm. The Fuzzballs also had a form of type-of-service priority queueing but, as with the Prue and Postel algorithm, allocations were not made on a user-by-user basis. The third policy is the Random-Dropping (RD) buffer management policy in which the service order is FCFS, but when the buffer is overloaded, the packet to be dropped is chosen at random [90]. This algorithm tends to allocate bandwidth to cooperative sources more or less evenly. However, it has been shown that the RD algorithm does not provide max-min fair bandwidth allocation, is vulnerable to ill-behaved sources, and is unable to provide reduced delay to conversations using less than their fair share of bandwidth [39, 56, 125, 154].

There are two objections that have been raised in conjunction with fair queueing. The first is that some source-destination pairs, such as file server or mail server pairs, need more than their fair share of bandwidth. This can achieved by weighted fair queueing, described below. Assign each source-destination pair a number n_α which represents how many queue slots that conversation gets in the bit-by-bit round robin. We now redefine N_{ac} as $N_{ac} = \sum n_\alpha$ with the sum over all active conversations, and P_i^α as $1/n_\alpha$ times the true packet length. With these changes, the earlier algorithm allocates each user a share of the bandwidth proportional to its weight. Of course, the truly vexing problem is the politics of assigning the n_α. Note that, while we have described an extension that provides for different relative shares of bandwidth, one could also define these shares as absolute fractions of the bandwidth of the outgoing line. This would guarantee a minimum level of service for these sources, and is very similar to the *Virtual Clock* algorithm of Zhang [154].

The other objection is that fair queueing requires the switches to be smart and fast. There is the technological question of whether or not one can build FQ switches that can match the bandwidth of fibers. If so, are these switches economically feasible? Work by Restrick and Kalmanek at AT&T Bell Laboratories has shown that it is possible to build fair queueing servers that switch ATM cells at 1.2 Gbps [69]. This indicates that building smarter switches does not necessarily have to make them slower.

Chapter 3: Efficient Implementation of Fair Queueing

3.1. Introduction

The performance of packet switched data networks is greatly influenced by the queue service discipline in routers and switches. While most current implementations are of the first-come-first-served discipline, Chapter 2 shows that the Fair Queueing (FQ) discipline provides better performance. Thus, there has been considerable interest in studying the theoretical and practical aspects of the algorithm [50, 57, 89, 93, 125, 126].

Chapter 2 discussed the properties of Fair Queueing; however, no particular implementation strategy was suggested. If future networks are to implement the discipline, it is necessary to study efficient implementation strategies. Thus, this chapter examines data structures and algorithms for the efficient implementation of Fair Queueing.

We begin by summarizing the Fair Queueing Algorithm. After pointing out its three components, we study the efficient implementation of each component. The component that critically affects the performance is a bounded size priority queue. In the rest of the chapter, we develop a technique to study average case performance of data structures, and use it to compare several priority queue implementations. Our results indicate that cheap and efficient implementations of Fair Queueing are possible. Specifically, if packet loss can be avoided, an ordered linked list implements a bounded size priority queue simply and efficiently. If losses can occur, then explicit per-conversation queues provide excellent performance.

3.2. The Fair Queueing algorithm

Let the *i*th packet from conversation α, of size P_i^α, arrive at a switch at time *t*. Let F^α denote the largest finish number for any packet that has ever been queued for conversation α at that switch. Then, we compute the packet's finish number F_i^α and the packet's bid number B_i^α as follows:

```
if( α is active )
        $F_i^\alpha = F^\alpha + P_i^\alpha$ ;
else
        $F_i^\alpha = R(t_i^\alpha) + P_i^\alpha$ ;
endif

$B_i^\alpha = P_i^\alpha + MAX( F^\alpha, R(t_i^\alpha) - \delta^\alpha)$ ;
$F^\alpha = F_i^\alpha$ ;
```

If the packet arrives when there is no more free buffer space, packets are dropped in order of decreasing bid number until there is space for the incoming packet. The next packet sent on the output line is the one with the smallest bid number.

3.3. Components of a Fair Queueing server

It is useful to trace a FQ server's actions on packet arrival and departure. When a packet arrives at the server, it first determines the packet's conversation ID α. The server then updates the current round number $R(t)$. The conversation ID is used to index into the server state to retrieve the conversation's finish number F^α and offset δ^α. These are used to compute the packet's finish and bid numbers, F_i^α and B_i^α.

If the output line is idle, the packet is sent out immediately, else it is buffered. If the buffers are full, some buffered packets may need to be discarded. On an interrupt from the output line indicating that the next packet can be sent, the packet in the buffer with the smallest bid number is retrieved and transmitted.

From this description, we identify three major components of a FQ implementation: bid number computation, round number computation, and packet buffering. We discuss each component in turn.

Bid number computation

A FQ server maintains, as its internal state, the finish number F^{α} and the offset δ^{α} of each conversation α passing through it. An implementor has to make two design choices: determining the ID of a conversation, and deciding how to access the state for that conversation.

The choice of the conversation ID depends on the entity to whom fair service is granted (see the discussion in Chapter 2), and the naming space of the network. For example, if the unit is a transport connection in the IP Internet, one such unique identifier is the tuple (source address, destination address, source port number, destination port number, protocol type). The elements of the tuple can be concatenated to produce a unique conversation ID. For virtual circuit based networks, the Virtual Circuit ID itself can be used as the conversation ID.

Note that for the IP Internet, one cannot always use the source and destination port numbers, since some protocols do not define them. For example, if an IP packet is generated by a transport protocol such as NetBlt [17], this information may not be available. An engineering decision could be to recognize port numbers for some common protocols and use the IP (source address, destination address) pair for all other protocols. This may result in some unfairness since transport connections sharing the same address pair would be treated first-come-first-served.

The conversation ID is used to access a data structure for storing state. Since IDs could span large address spaces, the standard solution is to hash the ID onto a index, and the technology for this is well known [65]. Recently, a simple and efficient hashing scheme that ignores hash collisions has been proposed [93]. In this approach, some conversations could share the same state, leading to unfair service, since these conversations are served first-come-first-served. However, this can be attenuated by occasionally perturbing the hash function, so that the set of conversations that share the same state changes periodically.

Round number computation

The round number at a time t is defined to be the number of rounds that a bit-by-bit round robin server would have completed at that time. To compute the round number, the FQ server keeps track of the number of active conversations $N_{ac}(t)$, since the round number grows at a rate that is inversely proportional to N_{ac}. However, this computation is complicated by the fact that determining whether or not a conversation is active is itself a function of the round number.

Consider the following example. Suppose that a packet P_0^A of size 100 bits arrives at time 0 on conversation A, and let $L = 1$. During the interval [0,50), since $N_{ac} = 1$, and $\partial R(t)/\partial t = 1/N_{ac}$, $R(50) = 50$. Suppose that a packet of size 100 bits arrives at conversation B at time 50. It will be assigned a finish number of 150 (= 50 + 100). At time 100, P_0^A has finished service. However, in the time interval [50, 100), $N_{ac} = 2$, and so $R(100) = 75$. Since $F^A = 100$, A is still active, and N_{ac} stays at 2. At $t = 200$, P_0^B completes service. What should $R(200)$ be? The number of conversations went down to 1 when $R(t) = 100$. This must have happened at $t = 150$, since $R(100) = 75$, and $\partial R(t)/\partial t = 1/2$. Thus, $R(200) = 100{+}50 = 150$.

Note that each conversation departure speeds up the $R(t)$, and this makes it more likely that some other conversation has become inactive. Thus, it is necessary to do an *iterative deletion* of conversations to compute $R(t)$, as shown in Figure 3.1.

The server maintains two state variables, t_{chk} and $R_{chk} = R(t_{chk)}$. A lower bound on $R(t)$ is $R_{chk} + L/N_{ac}(t_{chk})*(t - t_{chk})$, since N_{ac} is strictly non-increasing in [t_{chk}, t]. If some F^{α} is less than this expression, then conversation α has become inactive some time before time t. We determine the time when this happened, checkpoint the state at that time by updating the t_{chk}, R_{chk} pair, and repeat this computation till no more conversations are found to be inactive at time t.

Round number computation involves a MIN operation over the finish numbers, which suggests a simple scheme for efficient implementation. The finish numbers are maintained in a heap, and as packets arrive the heap is adjusted (since F^{α} is monotonically increasing for a given α, this is necessary for each incoming packet). This takes time O(log $N_{ac}(t)$) per operation. However, it only takes constant time to find the minimum, and so each step of the iterative

```
/* $F$, $\Delta$ and $N$ are temporary variables */
$N = N_{ac}(t_{chk})$;
do:
        $F = MIN(F^{\alpha} \mid \alpha \text{ is } active)$;
        $\Delta = t - t_{chk}$;
        if  $(F \le R_{chk} + \Delta * L/N)$ {
                declare the conversation with $F^{\alpha} = F$ inactive;
                $t_{chk} = t_{chk} + (F - R_{chk}) * N/L$;
                $R_{chk} = F$;
                $N = N - 1$;
        }
        else {
                $R(t) = R_{chk} + \Delta * L / N$;
                $R_{chk} = R(t)$;
                $t_{chk} = t$;
                $N_{ac}(t) = N$
                exit;
        }
od
```

Figure 3.1: Round number computation

deletion takes time O(log $N_{ac}(t)$) (for readjusting the heap after the deletion of the conversation with the smallest finish number).

In related work by Heybey *et al*, a heuristic for computing the round number has been proposed [57]. In this scheme, the round number is set to the finish number of the packet currently being transmitted, and all packets with the same finish number are served first-come-first-served. If this heuristic (or a small variant) is acceptable, the round number can be easily computed.

Packet buffering

FQ defines the packet selected for transmission to be the one with the smallest bid number. If all the buffers are full, the server drops the packet with the largest bid number (unlike the algorithm in Chapter 2, this buffer allocation policy accounts for differences in packet lengths). The abstract data structure required for packet buffering is a *bounded heap*. A bounded heap is named by its root, and contains a set of packets that are tagged by their bid number. It is associated with two operations, `insert(root, item, conversation_ID)` and `get_min(root)`, and a parameter, MAX, which is the maximum size of the heap.

`insert()` first places an item on the bounded heap. While the heap size exceeds MAX, it repeatedly discards the item with the largest tag value. We insert an item before removing the largest item since the inserted packet itself may be deleted, and it is easier to handle this case if the item is already in the heap. To allow this, we always keep enough free space in the buffer to accommodate a maximum sized packet. `get_min()` returns a pointer to the item with the smallest tag value and deletes it.

Determining a good implementation for a bounded heap is an interesting problem. There are two broad choices.

1 Since we are interested only in the minimum and maximum bid values, we can ignore the conversation ID, and place packets in a single homogeneous data structure.

2 We know that, within each conversation, the bid numbers are strictly monotonically increasing. This fact can be used to do some optimization.

It is not immediately apparent what the best course of action should be, particularly since per-conversation queueing is computationally more expensive. Thus, we did a performance analysis to help determine the best data structure and algorithms for packet buffering. The next sections describe some implementation alternatives, our evaluation methodology, and the results of the evaluation.

3.4. Buffering alternatives

We considered four buffering schemes: an ordered linked list (LINK), a binary tree (TREE), a double heap (HEAP), and a combination of per-conversation queueing and heaps (PERC). We expect that the reader is familiar with details of the list, tree and heap data structures. They are also described in standard texts such as References [58, 78].

Ordered list

Tag values usually increase with time, since bid numbers are strictly monotonic within each conversation (though not monotonic across conversations). This suggests that packets should be buffered in an ordered linked list, inserting incoming packets by linearly scanning from the largest tag value. Monotonicity implies that most insertions are near the end, and so this reduces the number of link traversals required.

Binary tree

We studied a binary tree, since this is simple to implement and has good average performance. Unfortunately, monotonic tag values can skew the tree heavily to one side, and the insertion time may become almost linear. This skew can be removed by using self-balancing trees such as AVL trees, 2-3 trees or Fibonacci trees. However, the performance of the self-balancing trees is comparable to that of a heap, since operations on balanced trees as well as heaps require a logarithmic number of steps. Since we do study heaps, we have not evaluated self-balancing trees explicitly. Our performance evaluation of heaps will also be representative of the results for self-balancing trees.

Double heap

A heap is a data structure that maintains a partial order. The tag value at any node is the largest (or smallest) of all the tags that lie in the subtree rooted at that node. Since we require both the minimum and the maximum elements in the heap, we maintain two heaps (implemented as arrays) and cross pointers between them. The code for implementing double heaps is presented in Appendix 1.

Per-conversation queue (PERC)

We know that, within a conversation, bid numbers are strictly monotonic. So, we queue packets per conversation, and keep two heaps keyed on the bid numbers of the head and tail of the queue for each conversation. `insert()` adds a packet to the end of the per channel queue and updates the max heap. `get_min()` finds the packet with smallest bid number from the min heap and dequeues it.

3.5. Performance evaluation

The performance of a data structure is measured by the cost of performing an elementary operation, such as an insertion or a deletion of an element, on it. Traditionally, performance has been measured by the asymptotic *worst* case cost of the operation as the size of the data structure grows without bound. For example, the insertion cost into an ordered list of length N is O(N), since in the worst case we may need to traverse N links to insert an item into the list.

How should we measure the performance of the four buffering data structures for the `insert()` and `get_min()` operations? Since constant work is needed to add or delete a single item at a known position to any data structure, the unit of work for linked lists and trees is a link traversal and for heaps is a swap of two elements. For linked lists and trees, the time for get_min( ) is a constant, and, for the other two data structures, it is comparable to the insertion time. Thus, an appropriate way to measure the performance of the data structures is to measure the number of links of the data structure that are traversed, or the expected number of swaps, during an insert( ) operation. Let B denote the number of buffers in a gateway, and let N denote the number of conversations present at any time (B is typically much larger than N). Table 3.1 presents well known results for the performance of the data structures described above for the insert( ) operation.

	Best	Worst	Average (Uniformly random workload)
LINK	O(1)	O(B)	O(B)
TREE	O(1)	O(B)	O(log(B))
HEAP	O(log(B))	O(log(B))	O(log(B))
PERC	O(log(N))	O(log(N))	O(log(N))

Table 3.1: Theoretical insertion costs

While the asymptotic worst case cost is an interesting metric, we feel that it is also desirable to know the average cost. However, average case behavior is influenced by the workload (the exact sequence of insert and get_min operations) that is presented to the data structure. Thus the best that we can do analytically is to assume that the workload is drawn from some standard distribution (uniform, gaussian, and so on), and compute the expected cost. We believe that this is not adequate. Instead, we use a general analysis methodology that we think is practical, and has considerable predictive power.

Methodology

We first parameterize the workload by some (small) number of parameters. Suitable values of the parameters are then fed to a realistic network simulator to create a trace of the workload for those parameter values. Then, we implement the data structure and associated algorithms, and measure the average performance over the trace length. This enables us to associate an average performance metric at that point in the workload parameter space. By judicious exploration of the parameter space, it is possible to map out the average performance as a function of the workload, and thus extrapolate performance to regions of the space that are not directly explored.

In our opinion, this methodology avoids a significant difficulty in average case analysis, that is, reliance on unwarranted assumptions about the workload distribution. Also, by mapping algorithm performance onto the workload space, it enables a network designer to choose an appropriate algorithm given the operating workload.

The drawback with this method is that it requires a realistic network simulator, and considerable amounts of computing time. Further, the parameterization of the workload and the exploration of the state space are more of an art than a science. However, we feel that these drawbacks are more than compensated for by the quality of the results that can be obtained.

Evaluation results

We chose the scenario of Figure 3.2 for detailed exploration. The gateway serves multiple sources (each of which generates one conversation) that share two common resources: the bandwidth of the output (trunk) line, and buffers in the gateway. Since there are no inter-trunk service dependencies, it suffices to model a single output trunk. Further, by changing the number of sources, and the number of buffers, it is possible to drive the system into congestion, something that we want to study. Finally, it is simple enough that it can be easily parameterized. Thus, our choice.

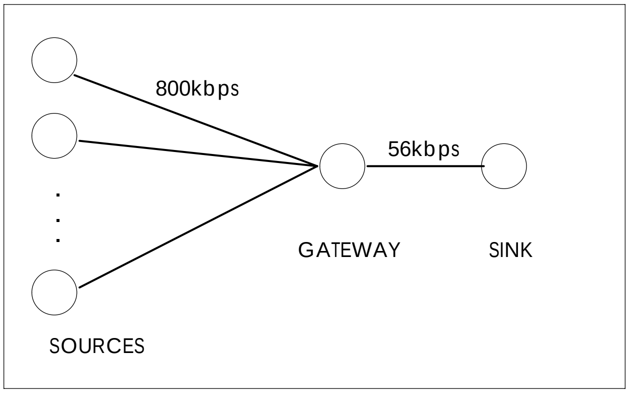
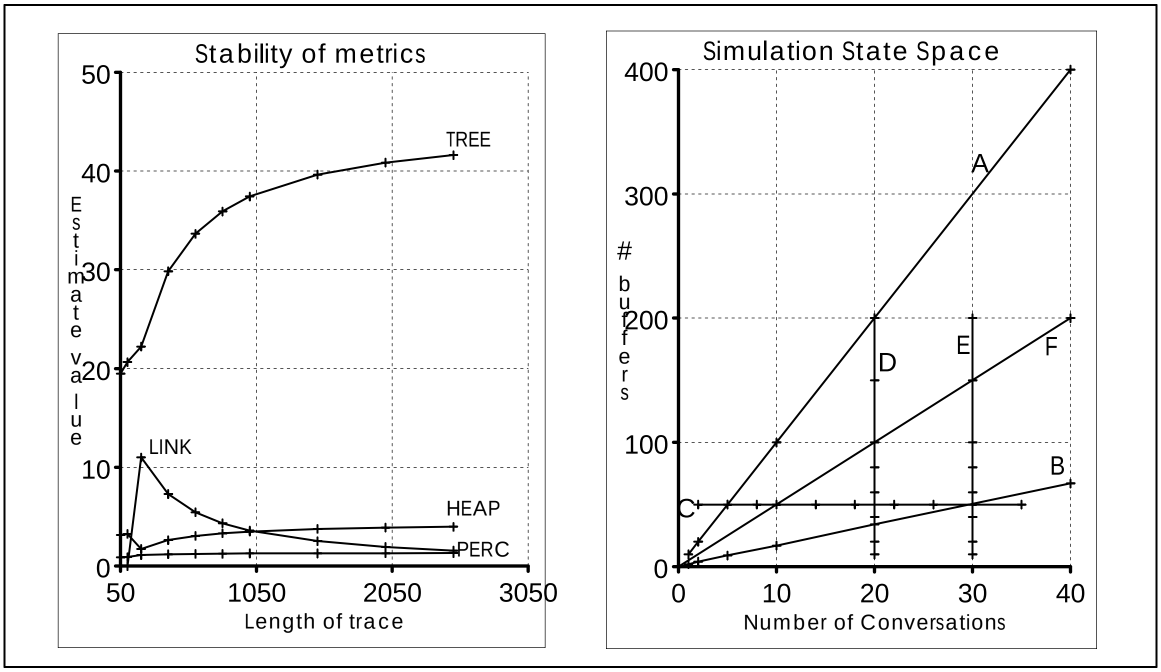


Figure 3.2: Simulation scenario



Figures 3.3 and 3.4

Note that we do not introduce any ‘non-conformant’ traffic in the sense of [57], since we wish to explore design decisions for well behaved sources only. If the network is expected to carry non-conformant traffic as well, then an evaluation of performance similar to the one described here needs to be carried out for that case.

The simulated sources obey the DARPA TCP protocol [108] with the modifications made by Jacobson [63]. They have a maximum window size of *W* each. By virtue of the flow control scheme, each source dynamically increases its window size, till either a packet is dropped by the gateway (leading to a timeout and a retransmission) or the window size reaches *W*. It is clear that the gateway cannot be congested if

$$W * N \leq B.$$

If the network is not congested, then each source behaves nearly independently, and the workload is regular, in the sense that the short term packet arrival and service rates are equal, and queues do not build up. When there is congestion, retransmissions and packet losses dramatically change the workload. Thus, one parameter that affects the workload is the ratio *N/B*. We also expect the workload to change as the number of conversations *N* increases. Thus, keeping *W* fixed, the two parameters that determine the workload are *N* and *B*.

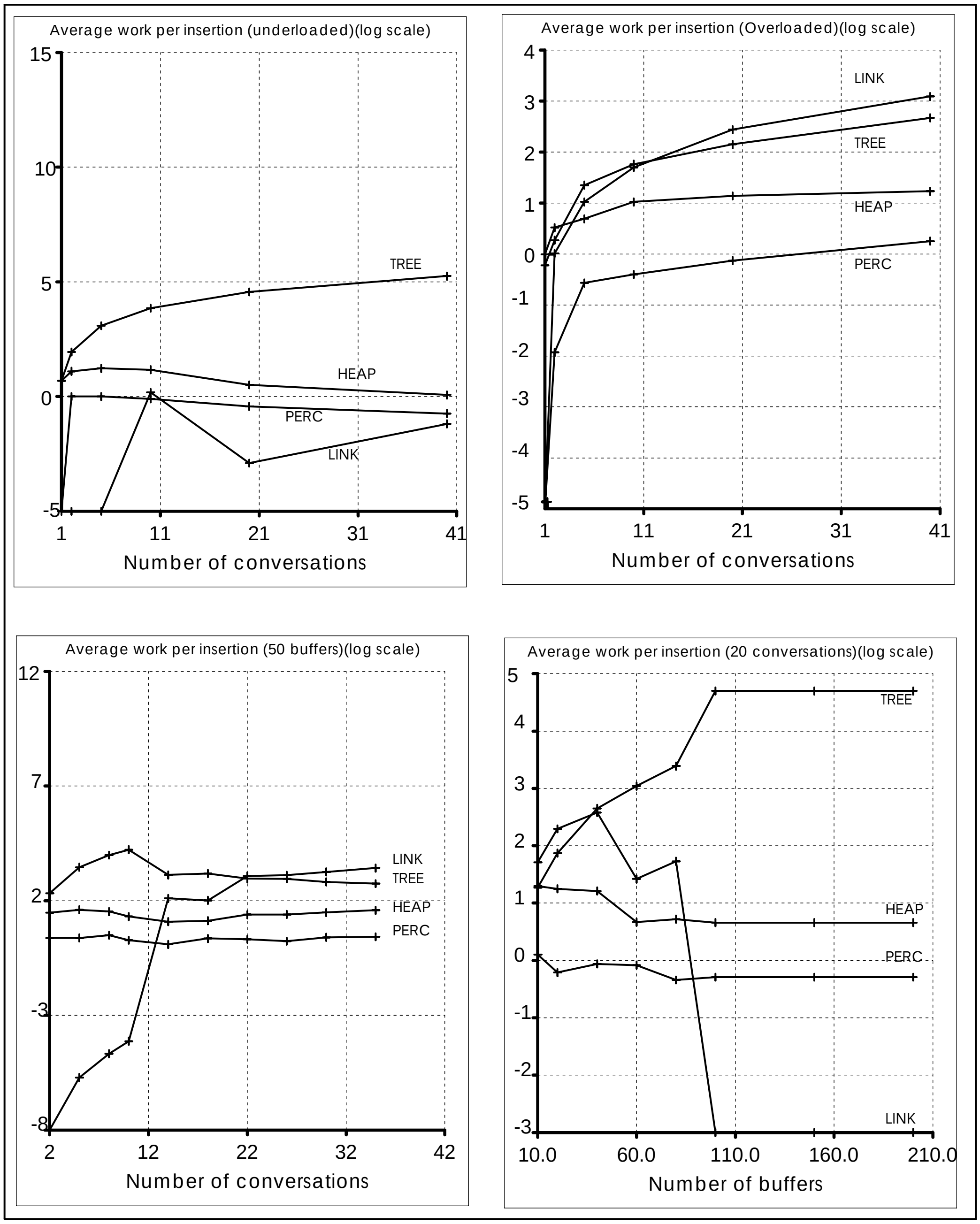


Figure 3.5: Average cost results

Following the experimental methodology outlined above, we used the REAL network simulator [73] to generate workload traces for a number of (*N*, *B*) tuples. One practical problem was to determine the appropriate trace length. Since generating a trace takes a considerable amount of computation, we decided to generate the shortest trace for which the cost metrics for all the four implementations stabilized. For simplicity, we determined this length for a single workload, with *N* = 10, *B* = 200, and generated a trace for 2500 seconds of simulated time. We then plotted the four cost metrics as a function of the trace length (Figure 3.3). We find that at 2500 seconds, all the metrics are no more than 10% away from their asymptotes. Since we only

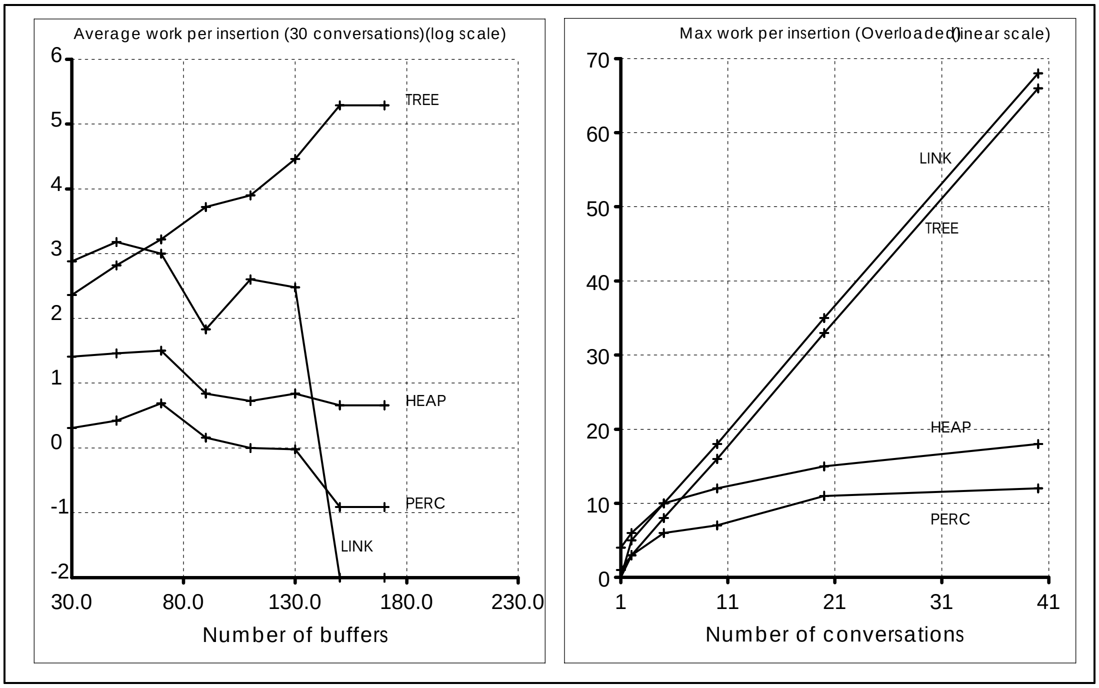


Figure 3.6: Average and maximum cost results

wanted to make qualitative cost comparisons, we generated each trace for 2500 seconds.

The (*N*, *B*) state space was explored along the five axes (labeled A through E) shown in Figure 3.4. Each '+' marks a simulation; there were a total of 35 simulations. Cost metrics for each implementation were determined along each axis. Axis A is the underloaded axis - along every point in the axis the gateway is lightly loaded, that is $W * N < B$. Symmetrically, axis B is the overloaded axis. Axes C, D and E are partly in the underloaded regime, and partly in the overloaded regime. Thus, congestion-dependent transitions in the relative costs of the implementations occur along these axes. The axis marked F is the locus of $W*N = B$.

Figures 3.5 and 3.6 show the average insertion cost along each of the five axes for each implementation. This is computed as

# elementary operations / # insertions in the trace

where an elementary operation is the traversal of a single link or a single heap exchange. All Y axes, though marked linearly, are drawn to logarithmic scale, so that, for example, 2 corresponds to e^2. Conceptually, one can imagine that for each implementation, there is a performance surface overlaying the workload space. Figures 3.5 and 3.6 represent cross sections of these surfaces as we slice along axes A-E. We can extrapolate the surfaces from these cross sections.

Results

Examination of the surfaces points out several facts:

- The performance surfaces for all the implementations (except LINK) are generally smooth, with few discontinuities. Thus, extrapolating the curves is meaningful.
- LINK behavior is somewhat erratic, since the insertion cost is is highly dependent on the workload. However, it still has a well defined behavior: in some cases, it is the by far the cheapest implementation, in others, it is clearly the most expensive. Figure 3.7 divides the

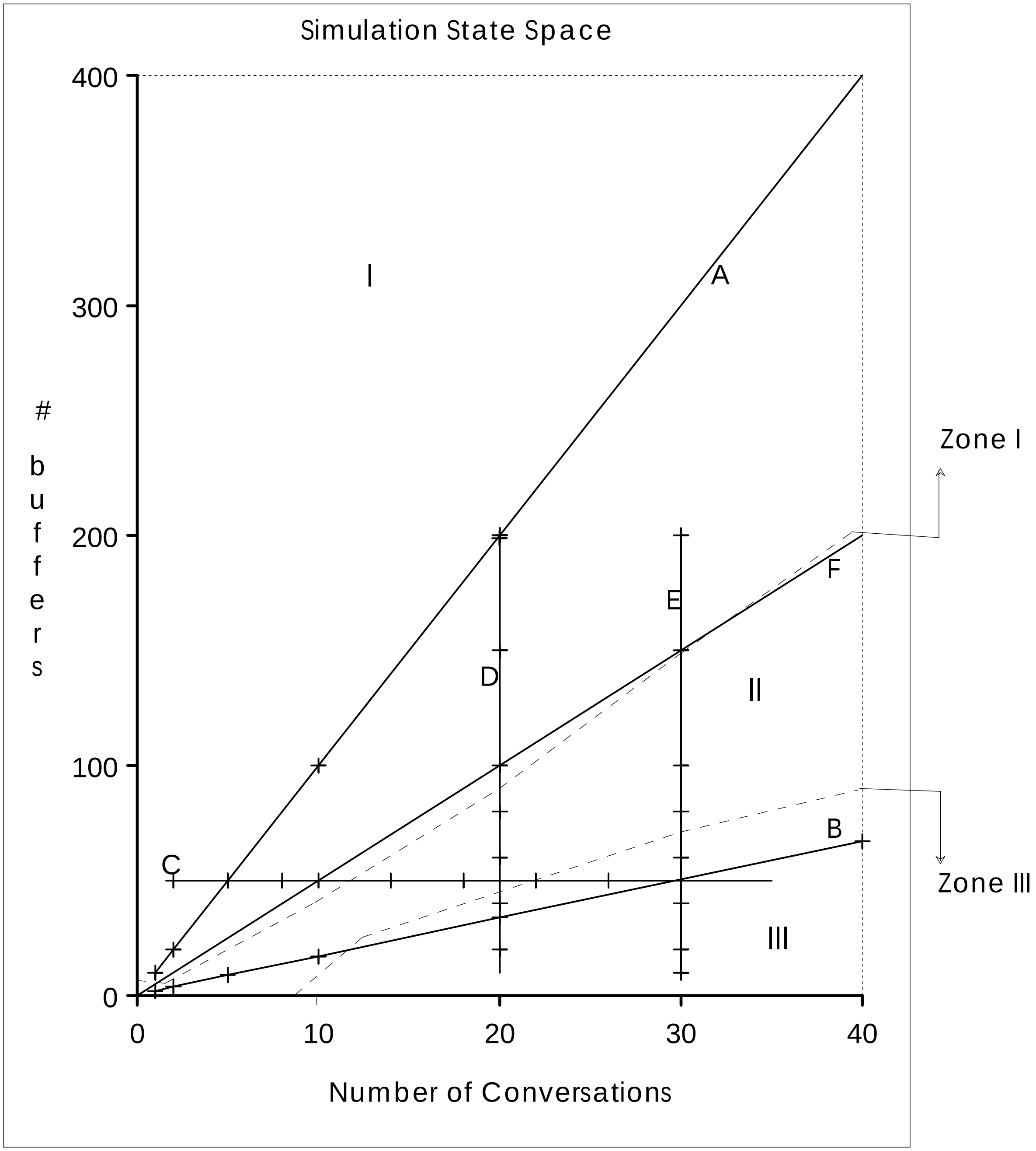


Figure 3.7: Linked list performance

workload space into three zones, numbered I-III. In zone I, it is best to use LINK, in zone III, LINK has the worst metric.

- As the number of conversations increases, the average HEAP and PERC insertion cost increases in the overloaded regime and is roughly constant in the underloaded regime.
- The cost metric for PERC is always less than that for HEAP or TREE.
- The cost metric for HEAP is within an order of magnitude of that for PERC in most cases.
- In the underloaded regime, binary trees become skewed, and hence are costly. They perform better in the overloaded regime.

- The average insertion cost for PERC is less than its theoretical average case cost.
- The maximum work done, which is shown for a typical case in Figure 3.6, is as expected in Table 3.1.
- In the underloaded case, HEAP and PERC show a declining trend, but this is offset by a larger increasing trend in the deletion time (not shown here).

Interpretation of results

The results give several guidelines for FQ implementation. TREE performs the worst in the underloaded regime; in the overloaded regime, HEAP and PERC are better. Hence, TREE is a bad implementation choice. We will not discuss it further.

Among the other strategies, PERC is always better than HEAP, and both of them have small worst case insertion costs. The worst case work per insertion is bounded by O(log(B)) for HEAP, and by O(log(N)) for PERC. Assuming that a gateway has 32 Mbytes of buffering per trunk line, and that packets are, on the average, 1Kbyte long, there will be at most on the order of 32K packets in the buffer. The number of conversations will be on the order of the square root of this number, i.e., around 200. With these figures, HEAP requires log(32K) ≈ 15, and PERC requires log(200) ≈ 8 elementary operations. Our simulations (Figure 3.5) show that in the trace driven simulation, the average work for HEAP and PERC is less than half of the worst case work. Thus, the average cost per insertion for PERC will be more like 4 elementary operations. This is a small price to pay to implement Fair Queueing.

The behavior of LINK (Figure 3.7) points to another implementation tactic. Note that in region I, LINK has the least cost. If the network designer can guarantee that the system will never enter the overloaded region (for example, by preallocating enough buffers for conversations, as in the Datakit network), then implementing LINK is the best strategy.

One consideration that is orthogonal to the insertion cost is implementation cost. For example, it is clear that implementing PERC involves much more work than implementing LINK. There are two implementation costs, corresponding to the work that is done independent of the number of elementary operations (static cost), and the work done per elementary operation (dynamic cost), respectively.

One simple metric to measure static cost is to measure the code size for `insert()`. We extracted the code for `insert()` and all the functions that it calls, for each implementation and placed it in a file. This file was compiled to produce optimized assembly code (in Unix, by the command cc -S -O -c). We then stripped the file of all assembler directives, leaving pure assembly code. Since this was done on a RISC machine, all instructions have the same cost, and the file length is a good metric of the complexity of implementing a given strategy. Table 3.2 presents this metric for the four implementations, normalized to the cost of implementing LINK.

Implementation	Static Cost	Dynamic Cost
LINK	1.0	5
TREE	1.1	18
HEAP	2.5	88
PERC	5.5	96

Table 3.2: Implementation cost

The dynamic cost was determined by examining the optimized assembly code, and counting the number of instructions executed per elementary operation. Table 3.2 presents the results. We did not specifically concentrate on reducing the number of instructions while writing the source code. We believe that the dynamic cost of the more expensive schemes can be considerably reduced by hand coding in assembly language.

To summarize, we draw four conclusions:

1 Implementing TREE is a bad idea.

2 HEAP provides good performance with low implementation cost.

3 PERC consistently provides the best performance, but has the highest implementation cost.

4 If the network designer can guarantee that the network never goes into overload, LINK is cheap to implement and has the minimum running cost.

3.6. Conclusions

In this chapter, we have considered the components of a FQ server, and have presented and compared several implementation strategies. Our work indicates that cheap and efficient implementations of FQ are possible. Along with the work done by McKenney [93] and Heybey et al [57], this work provides the practitioner with well defined guidelines for FQ implementation. We hope that these studies will encourage more implementations of Fair Queueing in real networks.

The performance evaluation methodology described here enables realistic evaluation of the average case performance of network algorithms. As LINK shows, this can lead to interesting results. We believe that a similar methodology can be used to evaluate a number of other workload sensitive network algorithms.

Finally, we believe that these results can be extended to other scheduling disciplines that are similar to Fair Queueing, such as the Virtual Clock algorithm [154]. Thus, our work has some generality of application.

3.7. Future work

This chapter does not examine hardware implementations of Fair Queueing. Given the need for faster packet processing in high speed networks, this is an obvious direction to pursue.

While we presented the means for the cost metric, we ignored the variance. This is because our simulations are completely deterministic. It would be useful to enhance the performance methodology described earlier to determine the variance and confidence intervals.

3.8. Appendix 3.A

A double heap consists of a pair of heaps. Since operations on one heap must be reflected in the other, we need pointers between the two instances of an element in the double heap. Since we represent heaps as arrays, pointers are indices, and we implement cross pointers using two integer arrays of indices.

The physical data structures used are four arrays, min, max, i_min, and i_max. min and max are the arrays that store the two heaps, one has the minimum element at the root, the other has the maximum. i_min[k] is the position in max of the kth element of min. i_max is defined symmetrically.

Every move in either heap must update i_min and i_max. We note that the only time an element is moved is when it is exchanged with some other element. We encapsulate this into an operation exchg() that swaps elements in the min or max heap, and simultaneously updates i_min and i_max so that the pointers are consistent. We actually need two symmetric operations, min_exchg() and max_exchg(), that swap elements in the min and max heap respectively. min_exchg() looks like the following:

```
min_exchg(a, b)      /* calls to swap are call by name */
{
swap(min[a], min[b]);
swap(i_max[i_min[a]], i_max[i_min[b]]);
swap(i_min[a], i_min[b]);
}
```

We now prove that this operation preserves pointer consistency, i.e. that i_min[i_max[a]] = a and i_max[i_min[a]] = a. Elements are inserted only in the last (say, nth) position in the heap, so the initial pointer positions are: i_min[n] = i_max[n] = n . It is easy to see that at the end of each min_exchg() operation, the pointers will remain consistent. Hence, by induction, pointers are always consistent.

Given the exchange operation, the rest of the heap operations are simple to implement. Heap insertion is done by placing data in the last element, and sifting up.

```
min_insert(data,num)
/* num is the current size of the heap */
{
ptr = num + 1;
min[ptr] = data;

for (; (ptr/2 >= 1) &&(min[ptr] < min (ptr/2]); ptr /=2)
    min_exchg(ptr, ptr/2);
}
```

Deletion is done by changing both the min and the max heaps, then adjusting them to recover the heap property.

```
min_delete()
{
int save;

min[1] = INFINITY;
save = i_min[1];
min_exchg(1,num);
min_adjust(1);

max[save] = -1;
max_exchg(save,num);
max_adjust(save);
}
```

Adjusting a heap consists of recursively sifting the marked element up or down as the case may be. Termination in a logarithmic number of steps is assured: because of the heap property, calls either go up the heap or down, and there can be no cycles.

```
min_adjust(a)
{
int smaller, smaller_son;

smaller = a;
if (min[a] < min[a/2]) smaller = a/2;
smaller_son = (min[lson(a)] < min[rson(a)]) ? lson(a) : rson(a);
if (min[smaller_son] < min[a]) smaller = smaller_son;
if (smaller != a)
    {
     min_exchg(a, smaller);
     min_adjust(smaller);        /* recursive call */
    }
}
```

Chapter 4: The Packet Pair Flow Control Protocol

4.1. Introduction

Most current packet switched data networks have routers and switches that obey a first-come-first-served (FCFS) queueing discipline, and existing transport layer flow control protocols have been optimized for such networks. If the service discipline is changed to Fair Queueing, then flow control protocols can improve their performance. This chapter presents the design and analysis of a flow control scheme, called the Packet-Pair flow control protocol, enabled by the Fair Queueing discipline. We first present a deterministic model for networks of Fair Queueing servers to motivate the design of Packet-pair. We then give implementation details. Subsequent sections deterministically analyze the transient behavior of Packet-pair.

4.2. Fair Queueing servers

Consider the queue service discipline in the output queues of packet routers. If packets are scheduled in strict Time-Division-Multiplexing (TDM) order, then whenever a conversation's time slot comes around and it has no data to send, the output trunk is kept idle and some bandwidth is wasted. Suppose packets are stamped with a priority index that corresponds to the packet's service time were the server actually TDM. It can be shown that service in order of increasing priority index approximately emulates TDM without its attendant inefficiencies [50]. This idea lies behind the Fair Queueing (FQ) service discipline.

With a FQ server, there are two reasons why the rate of service perceived by a specific conversation may change. First, the total number of conversations served can change. Since the service rate of the selected conversation is inversely proportional to the number of active conversations, the service rate of that conversation also changes.

Second, if some conversation has a low arrival rate, or has a bursty arrival pattern, then there are intervals where it does not have packets to send, and the FQ server treats that conversation as idle. Thus, the effective number of active conversations decreases, and the rate allocated to all the other conversations increases. When the traffic resumes, the service rate again decreases.

Note that even with these variations in the service rate, a FQ server provides a conversation with a more consistent service rate than a FCFS server. In a FCFS server the service rate of a conversation is linked in detail to the arrival pattern of every other conversation in the server, and so the perceived service rate varies rapidly.

For example, consider the situation where the number of conversations sending data to a server is fixed, and each conversation always has data to send when it is scheduled for service. In a FCFS server, if any one conversation sends a large burst of data, then the service rate of all the other conversations effectively drops until the burst has been served. In a FQ server, the other conversations will be unaffected. Thus, the server allocates a rate of service to each conversation that is, to a first approximation, independent of the conversations' arrival patterns. This motivates the use of a rate-based flow control scheme that determines the allocated service rate, and then sends data at this rate.

Choice of network model

We would like to design the flow control mechanism for a source in a network of FQ servers on a sound theoretical basis. This requires an analytic model for network transients. The standard network analysis technique is stochastic queueing analysis, where, for tractability, the usual assumptions are that the network consists of M/M/1 servers, the sources inject Poisson traffic, and the sources generate traffic independently. There are three problems with this approach. First, the strong assumptions regarding servers and sources are not always justifiable in practice. Second, the kind of results that can be obtained are those that hold in the average case, for example, expected queueing delays, and expected packet loss rates. Though the Chapman-Kolmogorov differential equations describe the exact dynamics (and thus the transient behavior) of a single M/M/1 queue, the solution of these equations is as hard as evaluating an infinite sum of Bessel functions [135]; besides, extending this analysis to a network of M/M/1 servers is difficult. Third, even if an exact analysis of transients in the network is obtained by an extension of the Chapman-Kolmogorov equations, if the servers are not M/M/1, no such differential equations are known.

Thus, using stochastic queueing analysis, transient analysis of a network of FQ servers (which are not M/M/1) is cumbersome, and perhaps impossible. However, flow control depends precisely on such transients. Thus, we prefer an approach that models network transients explicitly, but without these complications.

We model a single conversation in a network of FQ servers using deterministic queueing analysis. This model, formally defined in the next section, makes a major assumption that the service time per packet, defined as the time between consecutive packet services from a conversation, is assumed to be constant at each server. This is true if all the packets in a given conversation are of the same size and if the number of active conversations (conversations that have data to send when their turn in round-robin order comes by) at each FQ server is constant. The packet size assumption is borne out by studies of data traffic in current networks [13, 51], and will certainly hold in ATM networks of the near future. The other assumption is harder to justify. A FQ server isolates a conversation from other conversations if they are not too bursty, but this is not sufficient justification. We treat this assumption as a necessary crutch to aid deterministic analysis. We do not expect this assumption to hold in practice, and later in this chapter, the assumption is relaxed to allow infrequent, single sharp changes in the number of active conversations. However, note that in the important case of a network of FCFS servers, the deterministic service time assumption is wrong. Hence, for FCFS networks, our analysis is incorrect, and the Packet-pair flow control protocol is infeasible.

With these caveats in mind, it is nevertheless interesting that a deterministic modeling of a FQ server network, though naive, allows network transients to be calculated exactly [129]. Waclawsky and Agrawala have developed and analyzed a similar deterministic model for studying the effect of window flow control protocols on virtual circuit dynamics [144, 145].

Model

We model a conversation in a FQ network as a regular flow of packets over a series of servers (routers or switches) connected by links. The servers in the path of the conversation are numbered 1,2,3...n, and the source is numbered 0 (notations is summarized in the Appendix to this chapter). The source sends packets to a destination, and the destination is assumed to acknowledge each packet. (Strictly speaking, this assumption is not required, but we make it for ease of exposition.) We assume that sources always have data to send (an infinite-source assumption). This simplification allows us to ignore start-up transients in our analysis. The start-up costs can, in fact, be significant, and these are analyzed in [129]. However, for simplicity of exposition, we assume infinite sources from now on.

The service time at each server is deterministic. If the *i*th server is idle when a packet arrives, the time taken for service is s_i, and the (instantaneous) service rate is defined to be $\rho_i = 1/s_i$. Note that the time to serve one packet includes the time taken to serve packets from all other conversations in round-robin order. Thus, the service rate is the inverse of the time between consecutive packet services for the same conversation. The time taken to traverse a link is assumed to be zero (if it is not, it can always be added to the service time at the previous server).

If the server is not idle when a packet arrives, then the service time may be more than s_i. This is ignored in the model, but we consider its implications in a later section. If there are other packets from that conversation at the server, the packet waits for its turn to get service (we assume a FCFS queueing discipline for packets of the same conversation). We assume a work-conserving discipline, which implies that a server will never be idle whenever a packet is ready.

The source sending rate is denoted by ρ_0 and the source is assumed to send packets spaced exactly $s_0 = 1/\rho_0$ time units apart. We define

$$s_b = \max_i (s_i \mid 0 \le i \le n)$$

to be the *bottleneck* service time in the conversation, and b is the index of the bottleneck server. μ is defined to be $\frac{1}{s_b}$, and is the bottleneck service rate.

We now introduce the notion of a rate-throttle, by means of a recursive definition. To start with, the first server in the path of a conversation is a rate-throttle. Consider the servers along the path from the source to the destination. A server on the path is a rate-throttle if it is slower than some previous rate-throttle. Let SL, the ordered set of rate-throttles, be the set of strictly slower servers from the source to the bottleneck.

We now prove several lemmas about the properties of such conversations. Similar results and a more detailed analysis can be found in [143, 145].

Lemma 1 : (Basic lemma)

Consider data arriving at an initially idle server j at a rate r.

(a) If $r \le \rho_j$, there is no queueing at j, and the departure rate from server j is r.

(b) If $r > \rho_j$, there is queueing at j, and the departure rate from server j is ρ_j.

Proof :

(a) Initially, since the server is idle, its queue is empty. If the first packet arrives at time t_0, it will depart at time $t_0 + s_j$. Packets in the arriving stream are spaced $1/r$ time units apart. Thus, the next packet arrives at time $t_0 + 1/r$. Since $\rho_j \ge r$, $1/r \ge 1/\rho_j$ and $t_0 + 1/r \ge t_0 + s_j$, so the next packet arrives only after the first one has left. Thus, there is no queueing at the server. Simple induction on the sequence number of the arriving packet gives us the result on queueing.

The departure rate of the packets is constrained only by the arrival rate, and hence the output stream from the server has a rate r.

(b) Since the departure of the first packet happens after the arrival of the next packet, the second packet will be queued in the server. If there is a queue already, and a packet arrives before the departure of the previous packet, it will only add to the queue. Induction on the packet sequence number gives us the queueing result.

Since the departure stream from the server has an inter-packet spacing of s_j, the output stream is at rate ρ_j.

Lemma 2 : (Composition)

Consider two adjacent servers j and j+1. If data enters server j at a rate r such that $\rho_j \ge r > \rho_{j+1}$ queueing occurs only at server j+1.

Proof :

Since $r \le \rho_j$, there is no queueing at server j (Lemma 1). Hence, the departure rate of packets from server j, as well as the arrival rate at server j+1 is r. Since $r > \rho_{j+1}$, there is queueing at server j+1 (Lemma 1).

Lemma 3 : (Single bottleneck)

If data enters a segment of the VC numbered k, k+1, .. L, at a rate r such that $\rho_L < r < \rho$, $\rho \in \{\rho_k, \rho_{k+1}, \cdots, \rho_{L-1}\}$, then queueing occurs only at L.

Proof :

Since $r < \rho_k$, there is no queueing at server k and the departure rate from server k is r (Lemma 1). We can thus delete server k from the chain, and repeat the argument for the servers k+1, k+2, ... L. For the servers L-1, L, we use Lemma 2 to get the desired result.

Lemma 4 : (Chain of rate-throttles)

Queueing can happen only in elements of SL, the set of strictly slower servers.

Proof:

Break up the server chain 1,2, ... , b into sub-chains 1,2, ... sl_1; sl_1, sl_1+1, $\cdots$, sl_2; ... , such that only $sl_i \in$ SL. Consider the first such chain. If $\rho_0 < \rho_{sl_1}$, there is no queueing at sl_1. Hence, to get the worst possible scenario, we assume that $\rho_0 > \rho_{sl_1}$. In that case, from Lemma 3, the only queueing at the first chain will be at sl_1 (if ρ_0 is very large, sl_1 could just be 1). By definition of SL, the departure rate from sl_1, ρ_{sl_1}, satisfies the requirements for Lemma 3, so there will be queueing at sl_2, and at no other node in that subchain. From induction on the sequence number of the subchain, we get the desired result.

Lemma 5 : (Probing)

If a source sends packets spaced s_0 time units apart, and $\rho_0 \geq \rho_b$, the acks will be received at the source at intervals of s_b time units.

Proof :

By definition of the bottleneck, and Lemmas 1 and 4, the departure rate of packets at the bottleneck is μ. Since acks are created for each packet instantaneously, the acks will be spaced apart by s_b.

Define Δ_j to be $\rho_{sl_{j-1}} - \rho_{sl_j}$.

Lemma 6 : (Burst dynamics)

If a source sends a burst of K packets at a rate $s_0 \gg \rho_i$, for all i, then the queue at sl_j builds up at the rate Δ_j, reaches its peak at time $t_j = \sum_{i=0}^{j-1} s_i + \frac{K}{\rho_{sl_{j-1}}}$, and decays at the rate ρ_{sl_j}.

Proof :

Consider the situation at sl_j. This server receives packets at a rate $\rho_{sl_{j-1}}$, and serves them at the rate of ρ_{sl_j}. Thus, the queue builds up at the rate Δ_j. The queue reaches the maximum size when the last packet from the previous rate-throttle arrives. Since this is at a rate $\rho_{sl_{j-1}}$, the time to receive K packets is $K/\rho_{sl_{j-1}}$. To this we add $\sum_{i=0}^{j-1} s_i$, which is the time that the first packet arrived, to get the desired result. Finally, the queue will decay at the service rate of the rate-throttle, i.e., ρ_{sl_j}.

Note that in our model, it is not possible to have more than one bottleneck. While queueing may occur at more than one node, the service rate of the circuit is determined by the lowest indexed server with a service rate of μ, and this will be the bottleneck.

4.3. Rate probing schemes

How should we design a flow control scheme for a FQ network? Since the network allocates each conversation a service rate at its bottleneck server, a simple flow control scheme would be to *probe* the server to determine its current service rate for that conversation, and then send data at that rate (note that each conversation has its *own* bottleneck server). Sending it any slower would result in loss of throughput, and any faster would result in queueing at the bottleneck. Thus, it is clear that we should use a rate-based flow control scheme [17]. Note that rate-based flow control is explicitly enabled by FQ networks.

Rate based flow control

Our first attempt at designing a rate-based flow control scheme modified an idea described by Clark et al. for NETBLT [17], but as shown below, it was not successful. If a source sends data at a rate ρ_0, and receives acknowledgments at a rate ρ_b, then a reasonable control scheme is: if $\rho_0 > \rho_b$, decrease ρ_0, else increase it. The idea is that the rate at which acknowledgments are received is approximately the rate which the FQ server has allocated to the conversation. This should match the sending rate.

The increase and decrease policies are multiplicative, that is the algorithm is

$$\textbf{if } (\rho_0 > \rho_b) \textbf{ then } \rho_0 = \alpha\rho_0 \textbf{ else } \rho_0 = \beta\rho_0$$

where $\alpha < 1$ and $\beta > 1$. As the service rate changes, this adaptive scheme should converge on the new rate, and the system should stabilize at the correct rate.

However, there are four problems. First, a source cannot determine an increase in available capacity except by sending at a slightly increased rate and looking at the stream of acknowledgments (acks). Thus, a sudden large increase in the service rate can be adjusted for only after several round trip times. This is undesirable, particularly in high speed networks, where the bandwidth delay product can be large. Second, it takes a few round trip times to adjust to a sharp decrease in service rate. In the meantime, the bottleneck queue builds up. Third, after a decrease, the source sends at very nearly the service rate, so the built up queues never shrink, and the network becomes more prone to packet loss. Finally, the rate probe tends to push the network towards congestion, since the source always tries an increased sending rate, until the rate can no longer be supported. These problems point to a need for a better rate control algorithm, such as Packet-pair.

4.4. The Packet-pair scheme

Packet-pair is described in three stages. First, we motivate the algorithm. This is followed by a complete description and implementation details.

Motivation

Packet-pair is based on three observations:

(1) *The probing lemma allows a source to determine the bottleneck service rate by sending two packets at a rate faster than the bottleneck service rate, and measuring the inter-ack spacing.*

Consider a packet pair as it travels through the system, as shown in Figure 4.1. The figure presents a time diagram. Time increases down the vertical axis, and each axis represents a node along the path of a conversation. Lines from the source to the server show the transmission of a packet. The parallelograms represent two kinds of delays: the vertical sides are as long as the transmission delay (the packet size divided by the line capacity). The slope of the longer sides is proportional to the propagation delay. After a packet arrives, it may be queued for a while before it receives service. This is represented by the space between the horizontal dotted lines, such as *de*.

In the packet-pair scheme, the source emits two back-to-back packets (at time *s*). These are serviced by the bottleneck; by definition, the inter-packet service time is s_b, the service time at the bottleneck. Since the acks preserve this spacing, the source can measure the inter-ack spacing to estimate s_b.

We now consider possible sources of error in the estimate. Server 1 also spaces out the back-to-back packets, so can it affect the measurement of s_b? A moment's reflection reveals that, as long as the second packet in the pair arrives at the bottleneck before the bottleneck ends service for the first packet, there is no problem. If the packet does arrive after this time, then, by definition, server 1 itself is the bottleneck. Hence, the spacing out of packets at servers before the bottleneck server is of no consequence, and does not introduce errors into the scheme. Another detail that does not introduce error is that the first packet may arrive when the bottleneck server is serving other packets and may be delayed by a time interval such as *de*. Since this delay is shared by both packets in the pair, this does not affect the observation.

However, errors can be introduced by the fact that the acks may be spread out more (or less) than s_b due to differing queueing delays along the return path. In the figure note that the first ack has a net queueing delay of $ij + lm$, and the second has a zero queueing delay. This has the effect of reducing the estimate of s_b.

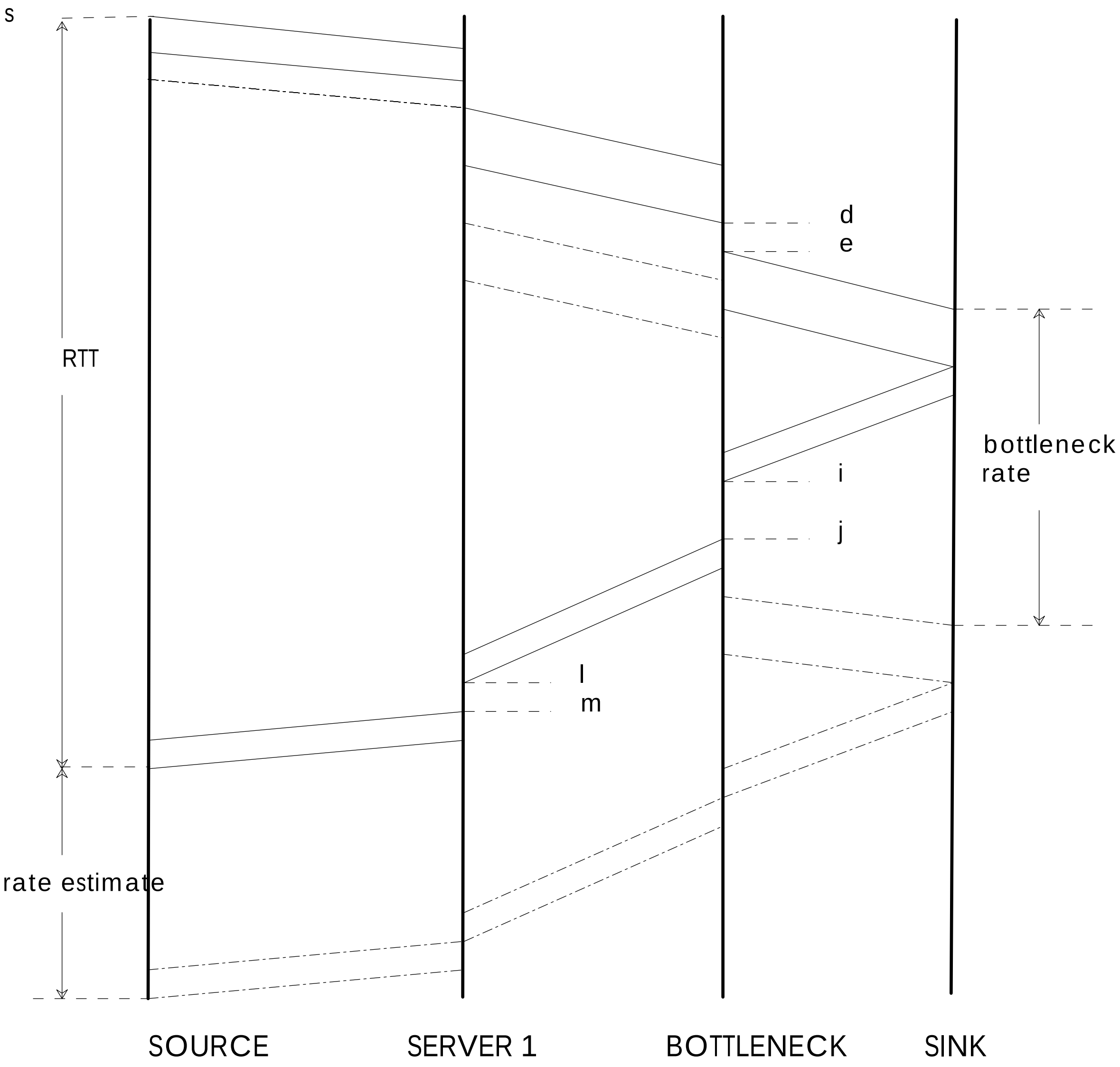


Figure 4.1: The packet-pair probing scheme

This source of error will persist even if the inter-ack spacing is noted at the sink and sent to the source using a state exchange protocol [120]. Measuring s_b at the sink will reduce the effect of noise, but cannot eliminate it, since any server that is after the bottleneck could also cause a perturbation in the measurement.

The conclusion is that the estimate of the service rate made by the sender can be corrupted by noise. In the deterministic model described earlier, even if the server is busy when a packet arrives, queueing delays are assumed to be zero, and thus Lemma 5 proves that the source observes the service rate exactly. In reality, these small queueing delays can cause observation noise. In a later section, we show how the flow control mechanism accounts for this.

(2) *If a source has a rate allocation $1/s_b$ and a round trip propagation delay R, it operates optimally when it has R/s_b packets outstanding.*

The packets sent from a source and not yet acknowledged constitute a pipeline, in the sense that they are being 'processed' in parallel by the network. Then V, the pipeline depth, is given by $V = R/s_b$. A source should keep exactly V packets outstanding to fully utilize the bottleneck bandwidth, and simultaneously have zero queueing delay [64, 97]. V depends on R and s_b. In the model presented earlier, the values of these quantities are fixed, but, in reality, they could change with time, and so it is necessary periodically to measure them. The source can measure s_b using the packet-pair method described above. R, the propagation delay, is the time between sending out a packet and receiving an ack when all the queues along the path are empty. This can be approximated by measuring r_t, the round trip time, though r_t will have a component due to the queueing delay.

(3) *If the pipeline depth V can increase or decrease by at most ΔV in any interval of time r_t, then keeping ΔV packets in the bottleneck queue's buffers will ensure that the bottleneck will not be idle.*

If an increase in R or s_b increases the pipeline depth by ΔV, some bottleneck bandwidth will be unutilized until the source reacts to the change. Since a source takes at least r_t time units to react, the source should have enough packets in the buffer to take up any transients. If the bottleneck queues ΔV packets, when V increases, the buffer will be drained, and no loss of throughput will occur. Thus, Packet-pair tries to ensure that, at any given time, at least ΔV packets are present in the bottleneck queue. We assume a buffer capacity of at least $2\Delta V$ per conversation at every switching node.

Note that this scheme avoids wasted bandwidth but adds a queueing delay (on average $\Delta V s_b$) to every packet served. A user can adjust the targetted bottleneck queue size to obtain a range of delay versus bandwidth loss tradeoffs. To get a lower average queueing delay, the bottleneck queue size should be kept small, but this introduces the possibility of a bandwidth loss when the pipeline depth increases. If this loss is to be avoided, then ΔV packets should be kept in the queue, but this will also increase the average queueing delay. We denote the target bottleneck queue size by n_b, and in the rest of the chapter, n_b is assumed to be ΔV. In practice, users who desire low queueing delays should choose n_b to be close to zero, while those who desire bulk throughput should choose a larger value. This is discussed in Chapter 5.

Algorithm

There are three phases in the operation of Packet-pair: start-up, queue priming and normal transmission.

At start-up, the source does not know the value of s_b. Since it should not overload the bottleneck with packets, some sort of 'slow-start' is desirable. This can be combined this with an initial measurement of the conversation parameters by sending a *packet-pair*, two packets sent as fast as possible (back-to-back). The round-trip time of the first packet gives us R_e, an estimator for R and the inter-arrival time of the two packets gives us s_e, an estimator for s_b.

Once the source computes $V_e = R_e / s_e$, an estimator of V, it can decide what n_b should be. n_b should be chosen depending on the value of ΔV for the network. This value can be determined empirically or the administrator can choose this to tune protocol performance. Deciding n_b a priori is possible, but not desirable, since an administrator might want n_b to be some fraction of V_e. Thus, the decision about the value of n_b is deferred till the end of the first round-trip-time. During queue priming, the source sends out a burst of n_b back-to-back packets so that the n_b packets accumulate in the bottleneck queue.

During normal transmission the source transmits packet-pairs every $2s_e$ time units and updates s_e based on the inter-arrival time between paired acks. R_e, the estimate for R, is updated to $r_t - n_b s_e$, which accounts for the queueing delay. To react immediately to changes in V, the source recomputes V_e on the arrival of every pair.

Let V_{new}, V_{old} be the new and old values of V_e using the new and old estimates, respectively. If $V_{new} < V_{old}$, the source calculates

$$n_{skip} = \max(\lceil (V_{old} - V_{new})/2 \rceil, 0),$$

where $\lceil z \rceil$ is the smallest integer greater than or equal to z. The source then skips n_{skip} transmission slots with a duration of the new value of s_e, and then continues to send pairs of packets at regular intervals of $2s_e$. If $V_{new} > V_{old}$, the source immediately transmits a burst of $V_{new} - V_{old}$ back-to-back packets (these are specially marked as non-paired packets).

Implementation

Figure 4.2 is the state diagram for a Packet-pair implementation.

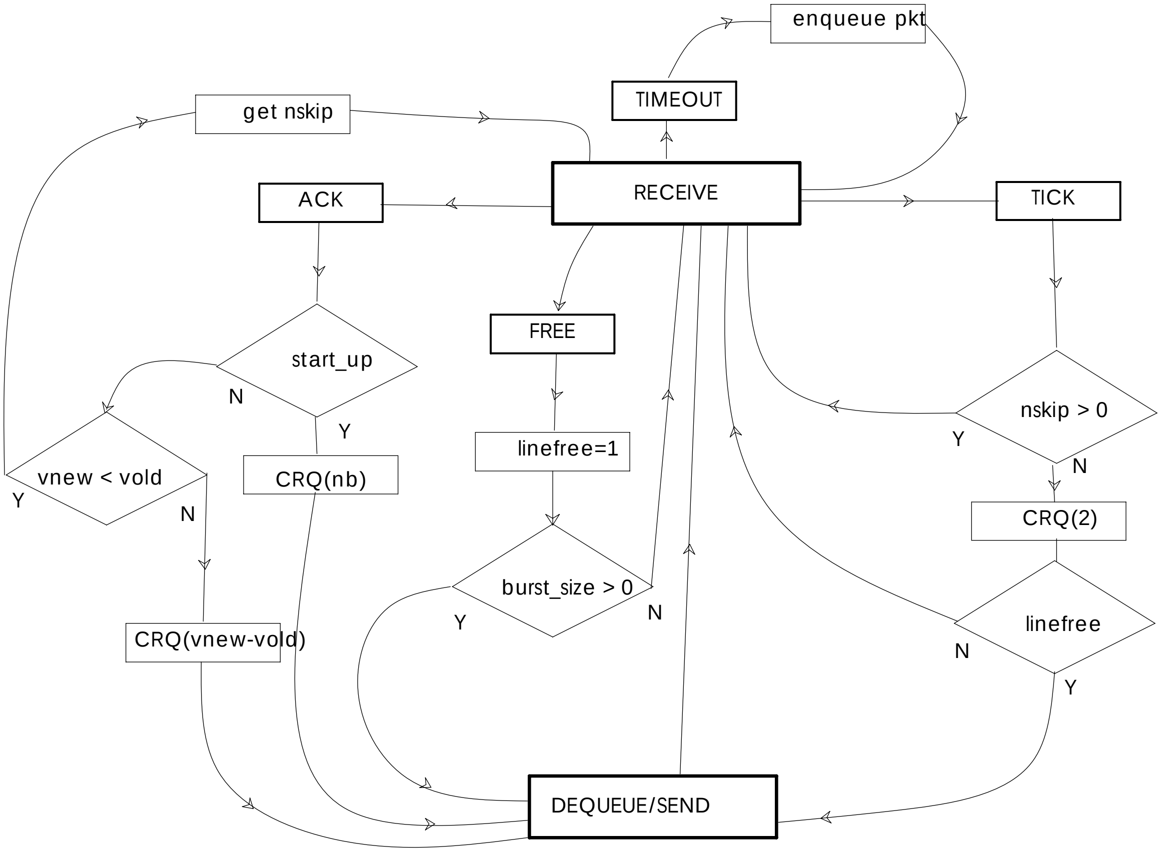


Figure 4.2: State diagram for implementing Packet-Pair
(CRQ(x) = accept x packets from user and enqueue)

A Packet-pair source usually is in the 'receive' state, waiting for an interrupt, one of

a) A signal indicating receipt of an acknowledgment packet. (ACK)
b) A 'tick' indicating that at the current sending rate, the next packet is due to be sent. (TICK)
c) A signal indicating that the output line is now free. (FREE)
d) A signal indicating that the last packet has timed out. (TIMEOUT)

There are two important state variables. `linefree` indicates that the output line from the source is free. `num_in_burst` is the number of packets enqueued in the output queue that belong to a burst. As long as num_in_burst is positive, a packet is dequeued and sent on the arrival of every FREE signal.

When an ACK signal arrives, if the ack is the first of a pair, R_e, the estimate for R, is updated. If it is the second of a pair, s_e is updated, and the source computes V_e and n_{skip}. If $V_{new} > V_{old}$, a burst of $V_{new} - V_{old}$ packets are queued on the output queue.

When an acknowledgment is received, some of the packets it acknowledges may have been timed out, and may be enqueued waiting to be sent out. So, at this point, all queued retransmissions that have become invalid are discarded. This implicitly assumes that retransmitted packets are queued at the transport layer. If a retransmitted packet has already been passed to a lower protocol layer, it will have to be retrieved from that layer, possibly violating layering. If this is a concern, this step can be ignored, since discarding retransmitted packets is not essential for the correct operation of the protocol.

If a TICK is received and n_{skip} is non-zero, it is decremented, the TICK timer is reloaded with $2s_e$, and we return to the receive state. Else, two packets (from the client) are enqueued on the output queue. If the line is free, one of the packets is dequeued and sent, else the source waits for an FREE to arrive. When an FREE arrives, if there are burst packets to be sent, one of them is dequeued and transmitted, else the source marks the line as free and returns to the receive state.

In current versions of TCP, there is a single retransmission timer, that is set on each packet transmission to $\hat{r}_t + 2MDM$ (ignoring some minor details). Here $\hat{r}_t$ is a moving average of the measured round trip time, and MDM is a moving average of the absolute difference between the measured round trip time and $\hat{r}_t$. When the timer goes off, the last unacknowledged packet is retransmitted. While we find this algorithm suitable, note that Packet-pair detects impending congestion using packet-pair probes, so, unlike TCP [63, 153], it is rather insensitive to the exact choice of the retransmission timer value. Thus, as a simplification, assuming one retransmission timer per packet, the retransmission timer value is simply Xr_t, where X is some small integer, and can be used as a tuning parameter (we used $X = 3$). On a TIMEOUT the timed out packet is placed in the output queue, waiting to be retransmitted. The new timeout value for the packet is twice the old value.

4.5. Analysis

We will analyze the the behavior of Packet-pair in the steady state (that is, when R and s_b do not change), and its response to transient changes in the virtual circuit. We will make four simplifying assumptions:

- Flow control is being done on behalf of an infinite source, that always has some data ready to send.
- Changes in V are bounded from above by ΔV, and the source knows or can estimate this value.
- Each server reserves $B \geq 2\Delta V$ buffers for each source.
- Transients are assumed to be due to a sharp, rather than a gradual, change in the system state. We assume that the value of a parameter, such as R, is constant until time t_0, at which point it changes discontinuously to its new value. We denote the value of $R(t)$ before the change as $R(t_0-\varepsilon)$, and after as $R(t_0+\varepsilon)$. We define $s_b(t_0-\varepsilon)$ and $s_b(t_0+\varepsilon)$ similarly.

Optimal flow control

We introduce the notion of optimal flow control and show that, in the steady state, Packet-pair is optimal. An optimal transmission flow control scheme should *always* operate at the knee of the load-throughput curve, so that maximum throughput is achieved with minimum delay [64]. As the conditions at the server change, the source flow control must adapt itself to the change. However, if we consider the speed-of-light propagation delay in this control loop for wide-area networks, it is clear that no realizable flow control scheme can always operate at the knee. Hence, we propose a weaker definition of optimality that is suitable for high throughput applications.

Let the bottleneck have B buffer spaces available for each source. Then, a flow control scheme is optimal in the interval $[T_0, T_1]$ if in every time interval $[t_1, t_2\ ] \in [\ T_0, T_1]$, there are no buffer overflows, and there is no loss of bandwidth at the bottleneck node. To be precise, at the bottleneck node, if the buffer occupancy at time t_1 is k,

$$0 \leq \int_{t_1}^{t_2} (\rho_0(t-d_1) - \rho_b(t))\ dt \ \leq\ B-k$$

where $\rho_0(t)$ is the source sending rate at time t, $\rho_b(t)$ is the bottleneck service rate at time t and d_1 is the propagation delay from the source to the bottleneck.

Steady state behavior of Packet-pair

In the steady state, Packet-pair will keep n_b packets in the bottleneck queue, and send packets at exactly the service rate, μ. Proposition 1 proves the optimality of Packet-pair in the steady state.

Proposition 1:

Let the transmission at the source start at time T_0 and end at time T_1. If V is constant in $(T_0, T_1]$, then Packet-pair is an optimal flow control scheme in $[T_0 + 2R(0), T_1\ ]$.

Proof: At time $T_0 + R(0)$, the source knows μ. Since $\Delta V = 0$, we can set $n_b = 0$, and priming the queue is not necessary. Thus, the source will immediately start to send a packet-pair every $2s_b$ time units. The first pair reaches the bottleneck at the latest by $T_0 + 2R(0)$. Since service is at rate μ, there is no build up of the queue. Clearly, no bandwidth is lost, and optimality conditions are trivially satisfied in $[T_0 + 2R(0), T_1\ ]$.

Remark

Note that many schemes described in the literature do not satisfy this weak notion of optimality even in the steady state. For example, the Jacobson-Karels version of TCP [63] drops packets if the maximum possible window size is larger than the buffer capacity at the bottleneck queue. The DECbit scheme keeps the queues at an average queue length of 1, and so will lose bandwidth when V increases [64]. As we mentioned earlier, NETBLT causes queues to build up whenever V decreases, and they are never adjusted for. Hence, a sequence of decreases in V will cause NETBLT to drop packets. Jain's delay based scheme [65] will respond poorly if V decreases due to a decrease in R, since it interprets the decrease in delay as a signal to increase the window size, which will cause further queueing, and possible packet loss. A more detailed analysis of these schemes can be found in [129].

Response to transients

In our model, the only network parameters visible to a source are μ and r_t. Thus, a flow control scheme can react to a change only in either of these variables, and we will study the response of Packet-pair to these changes. For each change, we study the packet or bandwidth loss, and the time taken to return to steady state.

Note that r_t itself depends on R and on the queueing delay in the bottleneck node. In steady state, queueing delay is constant, and r_t changes only if R or μ change. Thus, we need only consider changes in R and μ. In either case, the effect is to change $V = R.\mu$. We will denote the time at which the change occurred by t_0, and the change in V by $\delta V \leq \Delta V$.

4.5.1. Increase in propagation delay

4.5.1.1. Loss of bandwidth

R could increase if the VC is rerouted, and some new servers are added to the VC's path. There are two cases: either the path increase occurs before the bottleneck node, or after. If the increase happens after the bottleneck, then some server downstream of the bottleneck will have an increased idle time, and there is no loss in bottleneck bandwidth.

If the increase happens before the bottleneck, then for some period of time, the bottleneck could be idle, since packets that should have arrived at the bottleneck will now be sent to the new servers instead. Since we bound the increase in V by ΔV, if the bottleneck queue has ΔV packets, the buffered packets will be transmitted in the interim, and there will be no loss of bandwidth.

Another subtle possibility for bandwidth loss is if the first packet of a packet-pair reaches the bottleneck after time delay D_1 while the second one reaches there after a delay D_{1new}. The inter-arrival time of the pair at the bottleneck is then $\xi = D_{1new} - D_1$. If this is larger than s_b, the protocol will react by skipping some pairs of packets. However, Packet-pair will recover as soon as the next pair of acks arrive.

Proposition 2

The loss of bandwidth will be $2n + \xi / s_b$ packets, where

$$n = \max(\lceil \frac{1}{2} (\frac{R_{old}}{s_b} - \frac{R_{new}}{\xi}) \rceil ,0)$$

Proof:

The second term, ξ / s_b, is just the bubble size in the pipeline. Since Packet-pair mistakenly estimates that the pipeline depth is R_{new} / ξ, instead of R_{new} / s_b, it skips n slots, and in each slot it loses 2 packets, leading to a net loss of $2n$ packets.

4.5.1.2. Recovery time

Suppose R increased at time t_0. This information reaches the source at the latest by time $t_0 + R(t_0 + \varepsilon)$. The source will immediately send a burst of δV packets. Since the source is sending at rate μ, all but the bottleneck server will be idle when this burst is initiated. By the Burst dynamics lemma, we see that the last packet of the burst will arrive at the bottleneck at time $t_0 + \sum_{i=0}^{b-1} s_i + \frac{\delta V}{\rho_{sl_{b-1}}}$. This replenishes the bottleneck queue and so the steady state is regained. Subsequent increases in R are handled as before.

4.5.2. Increase in service rate

4.5.2.1. Loss of bandwidth

μ could increase if the number of conversations at the bottleneck decreases. The increase in μ could lead to some other node in the VC's path to become the bottleneck. We need to consider two cases, depending on whether the bottleneck migrates or not (that is, whether or not some other node becomes the bottleneck).

Assume that the bottleneck does not migrate. Due to the increase in μ, the bottleneck will serve packets faster than they arrive, and, until the first packet transmitted at the new rate arrives, there could be a loss of bandwidth. Now, the increase in μ becomes known to the source by time $t_0 + \sum_{i=b}^{n} s_i$, and the first packet from the burst reaches the server by $t_0 + R(t_0)$. So, if there are $(s_b(t_0 + \varepsilon) - s_b(t_0 - \varepsilon))R(t_0) \leq \Delta V$ packets in the bottleneck buffer, there will be no loss of bandwidth.

If the bottleneck migrates downstream from the old bottleneck, then packets queued at the old bottleneck will arrive at the new bottleneck and will form a queue there. It is easy to show that the only loss of throughput happens for the brief interval where the first packet from the old bottleneck is in transit to the new bottleneck.

If the bottleneck migrates upstream, then the δV packets queued at the old bottleneck cannot compensate for the bubble in the pipeline. The new bottleneck will be idle till the first packet from the compensatory burst arrive, and the loss could be as large as δV packets. Note that no flow control algorithm can prevent this loss, since the source must take at least $R(t_0)$ time to react to the change in s_b. Thus we have shown that no feasible flow control algorithm can satisfy even our weak notion of optimality in non-steady state operation.

4.5.2.2. Recovery time

The recovery time is just the time for δV packets to accumulate at the bottleneck, and this is the same as in the case of the increase in R, that is $\sum_{i=0}^{b-1} s_i + \frac{\delta V}{\rho_{L_{b-1}}}$.

4.5.3. Decrease in propagation delay

4.5.3.1. Loss of packets

R could decrease if packets are sent through a shorter route, but through the same bottleneck. We will assume that the packets in transit are not lost, but are received at the bottleneck. The effect of this change is to put an additional δV packets in the buffers of the bottleneck queue. Since in the steady state there are ΔV packets in the bottleneck, and we assume that we can buffer $2\Delta V$ packets, there is no packet loss.

4.5.3.2. Recovery time

The source knows the reduced value of R at the latest by time $t_0 + R(t_0 + \varepsilon)$. At this point, it will skip $(V_{old} - V_{new})/2$ transmission slots, taking a time equal to $1/2(R(t_0 - \varepsilon)/s_b - R(t_0 + \varepsilon)/sb)2sb$, $= R(t_0 - \varepsilon) - R(t_0 + \varepsilon)$. The first packet after resumption of normal transmission reaches the bottleneck latest by time $t_0 + R(t_0 - \varepsilon) - R(t_0 + \varepsilon) + R(t_0 - \varepsilon) = t_0 + R(t0 - \varepsilon)$. Since the excess packets accumulated at the bottleneck are exactly $(R(t_0 - \varepsilon)/s_b - R(t_0 + \varepsilon)/s_b)$, they will all be cleared in this time, and the system will reach the steady state at time $t_0 + R(t_0)$

4.5.4. Decrease in service rate

4.5.4.1. Loss of packets

The source knows of the decrease in μ at the latest by $t_0 + R(t_0)$, and will skip $\delta V/2$ transmission slots. The bottleneck could accumulate an additional δV packets in this time. Since there are $2\Delta V$ buffers, there is no packet loss.

4.5.4.2. Recovery time

The source detects the decrease in μ by time $t_0 + \sum_{i=b}^{n} s_i$, and will skip some transmissions. The first packet sent at the new rate is received at the bottleneck after a time $\sum_{i=1}^{b-1} s_i$, so that the first packet arrives at the bottleneck at time $t_0 + R(t_0) + skiptime$. During the first $R(t_0)$ time units, the bottleneck queue will accumulate packets in excess of ΔV, but exactly these many packets will be serviced during $skiptime$. Hence, at $t_0 + R(t_0) + skiptime$ the steady state is attained.

The source skips transmission for the time during which the bottleneck services the excess packets accumulated in its queue. Hence,

$$skiptime = \max(\lceil \frac{1}{2} \frac{R(t_0)}{s_b(t_0 - \varepsilon)} mi \frac{R(t_0)}{s_b(t_0 + \varepsilon)} \rceil, 0) sb(t_0 + \varepsilon)$$

4.6. Conclusions

This chapter models networks of FQ servers as a sequence of D/D/1 queues. In recent work [156], we have shown that the FQ discipline is similar to the Virtual Clock [154] and Delay-EDD [37] service disciplines. Thus, this modeling approach may be applied to networks of Virtual Clock and Delay-EDD servers as well. The network model initially assumes that the bottleneck service rate is constant. Later, this assumption is relaxed, and infrequent, single sharp changes in the bottleneck service rate are allowed. Some lemmas about the model are proved, which motivates the design of the Packet-pair flow control scheme. The detailed design of Packet-pair, as well as the state diagram of an implementation are presented. The Packet-pair protocol has several advantages over other flow control protocols: it responds quickly to changes in the network state, it takes advantage of FQ routers to probe network state, and it does not require

Change	Bandwidth loss	Packet loss	Recovery time
Increase in propagation delay	None	None	$\leq t_0 + R(t_0 + \varepsilon) + \sum_{i=0}^{b-1} s_i + \frac{\delta V}{\rho_{sl_{b-1}}}$
Increase in bottleneck service time	0, if no bottleneck migration $\leq \delta V$ if bottleneck migrates	None	As above
Decrease in propagation delay	None	None	$\leq t_0 + R(t_0)$
Decrease in bottleneck service time	None	None	$\leq t_0 + R(t_0) + s_b(t_0+\varepsilon)$ $\max(\lceil \frac{R(t_0)}{2s_b(t_0-\varepsilon)} - \frac{R(t_0)}{s_b(t_0+\varepsilon)} \rceil, 0)$

Table 4.1: Summary of analytic results

any assistance from routers, such as bit-setting. A similar approach to passively probing the network, though not using packet pairs, is described in reference [53], where each probe packet is time stamped, and the time series of delays suffered the probes is used to obtain a congestion indicator. Then, the packet sending rate is adjust to be proportional to the inverse of the level of congestion in the network. However, the Packet-pair protocol is not without its limitations. First, it requires that all the packet routers in the network implement Fair Queueing or a similar round-robin-like discipline. This is not necessary for flow control schemes such as the one in TCP [63, 109] or Jain's delay-based approach [65]. Since most current networks do not implement FQ, our approach is of limited practical significance, but it is hoped that this situation will change in the future. Second, Packet-pair assumes that changes in the bottleneck service rate happen slower than one round trip time. This is true if the conversations are long lived, and not too bursty. This is plausible if most of the traffic consists of fairly smooth (perhaps, uncompressed video) streams. However, if the number of active conversations can change drastically over the time scale of one round trip time, then Packet-pair is inadequate. We address this in Chapter 5, where we use a formal control-theoretic approach to flow control, and propose a hybrid flow control scheme that integrates window and rate-based flow control. Buffer reservations at each packet switch ensures that even if the flow control scheme incorrectly estimates the bottleneck service rate, there is no packet loss. With these changes, the packet-pair scheme performs well in FQ networks even if the bottleneck service rate changes rapidly and drastically.

4.7. Appendix 4.A : Notation

The following notation is used throughout the paper. Since a single conversation is studied, it is implicitly assumed that the variables are subscripted with the conversation identifier. The time dependencies are usually ignored in the text.

$s_i(t)$:	service time at the ith server in the path.
$\rho_i(t)$:	service rate at the i server, $\rho_i(t) = 1/s_i(t)$.
$s_b(t)$:	service time at the bottleneck server.
$\mu(t)$:	bottleneck service rate = $1/s_b(t)$.
$s_e(t)$:	estimator for $s_b(t)$.
sl_i:	ith rate throttle in the path.
Δ_j:	$\rho_{sl_{j-1}} - \rho_{sl_j}$
$R(t)$:	round trip propagation delay (excluding queueing delays).
$r_t(t)$:	$R(t)$ + queueing delay
$R_e(t)$:	estimator for R.
$V(t)$:	pipeline depth = R/s_b.
$V_e(t)$:	estimator for $V = R_e/s_e$.
δV:	actual change in V.
$\rho_0(t)$:	source sending rate
$R(t_0-\varepsilon)$:	$R(t)$ before a change at time t_0.
$R(t_0+\varepsilon)$:	$R(t)$ after a change at time t_0.
$s_b(t_0-\varepsilon)$:	$s_b(t)$ before a change at time t_0
$s_b(t_0+\varepsilon)$:	$s_b(t)$ after a change at time t_0
n_b:	desired number of packets in the bottleneck buffer

Chapter 5: A Control-Theoretic Approach to Flow Control

5.1. Introduction

If networks implement Fair Queueing at each server, then new flow control protocols are enabled. In Chapter 4, we presented the 2P protocol, which was enabled in this way. The 2P protocol is strongly motivated by a deterministic model for the network. Since, in practice, this assumption cannot always be justified, this chapter presents the design of a flow control protocol that works well even in the presence of stochastic changes in the network state. We use a control-theoretic approach to determine how a conversation can satisfy its throughput and queueing delay requirements by adapting its data transfer rate to changes in network state, and to prove that such adaptations do not lead to instability.

A control-theoretic approach to flow control requires that changes in the network state be observable. We have shown in Chapter 4 that it is possible to measure network state easily if the servers at the output queues of the switches do Fair Queueing, and the transport protocol uses the Packet-Pair probing technique. Thus, in this chapter, we will make the assumption that the queue service discipline is FQ and that the sources implement Packet-Pair. Our approach does not extend to First-Come-First-Served (FCFS) networks, where there is no simple way to probe the network state.

The chapter is laid out as follows. We first present a stochastic model for FQ networks (§5.2). Next, we use the Packet-Pair state probing technique as the basis for the design of a stable rate-based flow control scheme (§5.3). A problem with non-linearity in the system is discussed in §5.4. We present a Kalman state estimator in §5.5. However, this estimator is impractical, and so we have designed a novel estimation scheme based on fuzzy logic (§5.6). A technique to increase the frequency of control based on additional information from the system is presented in §5.7, and this serves as the basis for a new, stable control law. Practical implementation issues are discussed in §5.8, and these include correcting for parameter drift, and interaction with window flow control. We conclude with some remarks on the limitations of the approach (§5.9) and a review of related work (§5.10).

5.2. Stochastic model

In the deterministic model of Chapter 4, μ, the service rate at the bottleneck of a conversation, is assumed to be constant. Actually, μ changes due to the creation and deletion of active conversations. If the number of active conversations, N_{ac}, is large, we expect that the change in N_{ac} in one time interval will be small compared to the value of N_{ac}. Hence the change in μ in one interval will be small, and $\mu(k+1)$ will be 'close' to $\mu(k)$. One way to represent this would be for μ to be a fluctuation around a nominal value μ_0. However, this does not adequately capture the dynamics of the process, since $\mu(k+1)$ is 'close' to $\mu(k)$ and not to a fixed value μ_0. Instead, we model μ as a random walk where the step is a random variable that has zero mean and low variance. Thus, for the most part, changes are small, but we do not rule out the possibility of a sudden large change. This model is simple, and, though it represents only the first order dynamics, we feel that it is sufficient for our purpose. Thus, we define

$$\mu(k+1) = \mu(k) + \omega(k),$$

where $\omega(k)$ is a random variable that represents zero-mean gaussian white noise. There is a problem here: when μ is small, the possibility of an increase is larger than the possibility of a decrease. Hence, at this point, the distribution of ω should be asymmetric, with a bias towards positive values (making the distribution non-gaussian). However, if μ is sufficiently far away from 0, then the assumption of zero mean is justifiable.

The white noise assumption means that the changes in service rate at time k and time $k+1$ are uncorrelated. Since the changes in the service rate are due to the effect of uncorrelated input traffic, we think that this assumption is valid. However, the gaussian assumption is harder to justify. As mentioned in [2], many noise sources in nature are gaussian. Second, a good rule of thumb is that the gaussian assumption will reflect at least the first order dynamics of any noise distribution. Finally, for any reasonably simple control-theoretic formulation (using Kalman estimation), the gaussian white noise assumption is unavoidable. Thus, for these three reasons, we will assume that the noise is gaussian.

These strict assumptions about the system noise are necessary mainly for doing Kalman estimation. We also describe a fuzzy prediction approach (§5.6) that does not require any of these assumptions

Note that the queueing-theoretic approach to modeling μ would be to define the density function of μ, say $G(\mu)$, which would have to be supplied by the system administrator. Then, system performance would be given by expectations on the distribution. For example, a metric $X(\mu)$, that depends on the service rate μ would be described by $E(X(\mu)) = X(\hat{\mu})$, where $\hat{\mu}$, the average value of μ is given by $\hat{\mu} = \int \mu G(\mu)\, d\mu$. Now, if $G(\mu)$ is unknown, so is $\hat{\mu}$. Further, $E(X)$ depends on $\hat{\mu}$, an asymptotic average. In contrast, we explicitly model the dynamics of μ and so our control scheme can depend on the currently measured value of μ, instead of an asymptotic time average.

5.3. Design strategy

This section describes the strategy used to design the flow-control mechanism, some preliminary considerations, and the detailed design. The design strategy for the flow control mechanism is based upon the Separation Theorem [3]. Informally, the theorem states that, for a linear stochastic system where an observer is used to estimate the system state, the eigenvalues of the state estimator and of the controller are separate. The theorem allows us to use any technique for state estimation, and then implement control using the estimated state $\hat{x}$ instead of the actual state x. Thus, we will derive a control law assuming that all required estimators are available; the estimators are derived in a subsequent section. We first discuss our assumptions and a few preliminary considerations.

5.3.1. Choice of setpoint

The aim of the control is to maintain the number of packets in the bottleneck queue, n_b, at a desired setpoint. Since the system has delay components, it is not possible for the control to stay at the setpoint at all times. Instead, the system will oscillate around the setpoint value. The choice of the setpoint reflects a tradeoff between mean packet delay, packet loss and bandwidth loss (which is the bandwidth a conversation loses because it has no data to send when it is eligible for service). This is discussed below.

Let B denote the number of buffers a switch allocates per conversation (in general, this may vary with time; in our work, we assume that B is static). Consider the distribution of n_b for the controlled system, given by $N(x) = Pr(n_b = x)$ (strictly speaking, $N(x)$ is a Lebesgue measure, since we will use it to denote point probabilities). $N(x)$ is sharply delimited on the left by 0 and on the right by B, and tells us three things:

1) Pr(loss of bandwidth) = Pr (FQ server schedules the conversation for service | $n_b = 0$). Assuming that these events are independent, which is a reasonable assumption, we find that Pr(loss of bandwidth) is proportional to $N(0)$.

2) Similarly, Pr (loss of packet) = Pr (packet arrival | $n_b = B$), so that the density at B, $N(B)$, is proportional to the probability of a packet loss.

3) The mean queuing delay is given by

$$\frac{s_b}{\int_0^B N(x)\,dx} \int_0^B x N(x)\,dx,$$

where, on average, a packet takes s_b units of time to get service at the bottleneck.

If the setpoint is small, then the distribution is driven towards the left, the probability of bandwidth loss increases, the mean packet delay is decreased, and the probability of packet loss is decreased. Thus, we trade off bandwidth loss for lower mean delay and packet loss. Similarly, if we choose a large setpoint, we will trade off packet loss for a larger mean delay and lower probability of bandwidth loss. In the sequel, we assume a setpoint of $B/2$. The justification is that, since the system noise is symmetric, and the control tracks the system noise, we expect $N(x)$ to be symmetric around the setpoint. In that case, a setpoint of $B/2$ balances the two tradeoffs. Of course, any other setpoint can be chosen with no loss of generality.

Queueing-theoretic choice of setpoint

In recent work, Mitra et al [96, 97] have studied asymptotically optimal choices of window size for window based flow control, when the scheduling discipline at a switch is either FCFS or processor sharing (PS). With some caveats as to its generality, their approach is complementary to ours, and can provide insight into the choice of setpoint.

The basis for their study is product-form queueing network theory, where asymptotic network behavior is studied as the bandwidth-delay product tends to infinity. In this regime, the analysis of virtual circuit dynamics indicates that the network behaves almost deterministically (which reinforces our earlier claim). Further, if a source wishes to optimize the throughput to queueing delay ratio, or power, then the optimal window size K is given by

$$K = \lambda + \alpha\sqrt{\lambda}$$

where λ is the bandwidth delay product, and α is approximately $\frac{1}{2\sqrt{M}}$, M being the number of hops over which the circuit sends data. Thus, the optimal choice of the setpoint is simply $\alpha\sqrt{\lambda}$. Since α can be precomputed for a circuit, and changes in λ can be determined using the packet-pair protocol, the setpoint can be dynamically adjusted to deliver maximum power using the earlier control model. Thus, the two theoretical approaches can be used in conjunction to determine the bandwidth delay product, the optimal setpoint, and, also, a mechanism to keep the system at the optimal setpoint.

However, there are several difficulties with the queueing-theoretic approach that limit its generality. First, the analysis assumes that the scheduling discipline is either FCFS or PS. While Fair Queueing can be approximated by Head-of-Line Processor Sharing [50], approximating it by PS does not seem to be reasonable. Second, the analysis assumes that the cross traffic is strictly Poisson. Since there is no empirical evidence that the assumption is valid, it may be inaccurate. Third, the packet service time is assumed to be exponentially distributed. This assumption is almost certainly incorrect, since numerous studies have indicated that the packet size distribution is usually strongly multi-modal (usually bi-modal), so that the service times will follow a similar distribution [13, 51].

Nevertheless, even with these caveats, the queueing approach is a strong basis to justify our intuitions, and, to a first approximation, these results can be used to determine the choice of the setpoint in the control system.

5.3.2. Frequency of control

We initially restrict control actions to only once per round trip time (RTT) (this restriction is removed in §5.7). For the purpose of exposition, we divide time into *epochs* of length RTT (= R + queueing delays) (Figure 5.1). This is done simply by transmitting a specially marked packet-pair, and when it returns, taking control action, and sending out another marked pair. Thus, a control action is taken at the end of every epoch.

5.3.3. Assumptions regarding round trip time delay

We assume that the propagation delay, R, is constant for a conversation. This is usually true, since the propagation delay is due to the speed of light in the fiber and the hardware switching delays. These are fixed, except for rare rerouting.

We assume that the round trip time is large compared to the spacing between the acknowledgments. Hence, in the analysis, we treat the arrival of the packet pair as a single event, which measures both the round trip time and the bottleneck service rate.

Finally, we assume that the measured round trip time in epoch k, denoted by $RTT(k)$, is a good estimate for the round trip time in epoch k+1. The justification is that, when the system is in equilibrium, the queue lengths are expected to be approximately the same in successive epochs. In any case, for wide area networks, the propagation delay will be much larger than the additional delay caused by a change in the queueing delay. Hence, to a first approximation, this change can be ignored. This assumption is removed in §5.7.

5.3.4. Controller design

Consider the situation at the end of the kth epoch. At this time we know $RTT(k)$, the round trip time in the kth epoch, and $S(k)$, the number of packets outstanding at that time. We also *predict* $\hat{\mu}(k+1)$, which is the estimator for the average service rate during the (k+1)th epoch. If the service rate is 'bursty', then using a time average for μ may lead to problems. For example, if the average value for μ is large, but during the first part of the control cycle the actual value is low, then the bottleneck buffers could overflow. In such cases, we can take control action with the arrival of every probe, as discussed in §5.7.

Figure 5.1 shows the time diagram for the control. The vertical axis on the left represents the time of the source, and the axis on the right that of the bottleneck. Each line between the axes represents a packet pair. Control epochs are marked for the source and the bottleneck. Note that the epochs at the bottleneck are time delayed with respect to those at the source. We use the convention that the end of the kth epoch is called 'time k', except that $n_b(k)$ refers to the number of packets in the bottleneck at the *beginning* of the kth epoch.

We now make a few observations regarding Figure 5.1. The distance ab is the RTT measured by the source (from the time the first packet in the pair is sent to the time the first ack is received). By an earlier assumption, the propagation delay for the (k+1)th special pair is the same as for the kth pair. Then $ab = cd$, and the length of epoch k at the source and at the bottleneck will be the same, and equal to $RTT(k)$.

At the time marked 'NOW', which is the end of the kth epoch, all the packets sent in epoch k–1 have been acknowledged. So, the only unacknowledged packets are those sent during the kth epoch itself, and this is the same as the number of outstanding packets $S(k)$. This can be approximated by the sending rate multiplied by the sending interval, $\lambda(k)RTT(k)$. So,

$$S(k) = \lambda(k)RTT(k) \qquad 5.1$$

The number of the conversation's packets in the bottleneck at the beginning of the (k+1)th epoch is simply the number of packets at the beginning of the kth epoch plus what came in minus what went out in the kth epoch (ignoring the non-linearity at $n_b = 0$, discussed in §5.5.6). Since $\lambda(k)$ packets were sent in, and $\mu(k)RTT(k)$ packets were serviced in this interval, we have

$$n_b(k+1) = n_b(k) + \lambda(k)RTT(k) - \mu(k)RTT(k) \qquad 5.2$$

Equations (5.1) and (5.2) are the fundamental equations in this analysis. They can be combined to give

$$n_b(k+1) = n_b(k) + S(k) - \mu(k)RTT(k) \qquad 5.3$$

Now, $n_b(k+1)$ is already determined by what we sent in the kth epoch, so there is no way to control it. Instead, we will try to control $n_b(k+2)$. We have

$$n_b(k+2) = n_b(k+1) + (\lambda(k+1) - \mu(k+1))RTT(k+1) \qquad 5.4$$

From (5.3) and (5.4):

$$\begin{gathered} n_b(k+2) = n_b(k) + S(k) - \mu(k)RTT(k) + \qquad 5.5 \\ \lambda(k+1)RTT(k+1) - \mu(k+1)RTT(k+1) \end{gathered}$$

The control should set this to $B/2$. So, set (5.5) to $B/2$, and obtain $\lambda(k+1)$.

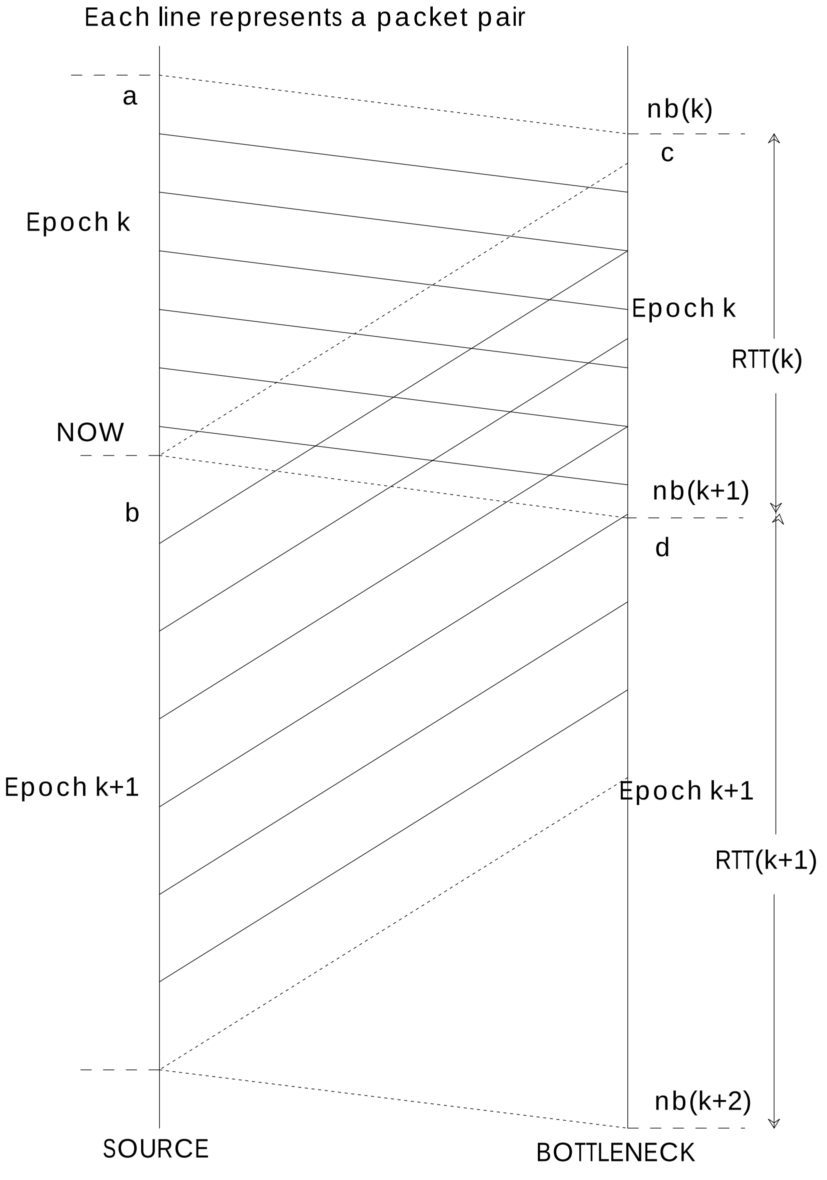


Figure 5.1: Time scale of control

$$n_b(k+2) = B/2 = n_b(k) + S(k) - \mu(k)RTT(k) + (\lambda(k+1) - \mu(k+1))RTT(k+1) \qquad 5.6$$

This gives $\lambda(k+1)$ as

$$\lambda(k+1) = \frac{1}{RTT(k+1)} [B/2 - n_b(k) - S(k) + \mu(k)RTT(k) + \mu(k+1)RTT(k+1)] \qquad 5.7$$

Replacing the values by their estimators (which will be derived later), we have

$$\lambda(k+1) = \frac{1}{\hat{RTT}(k+1)} [B/2 - \hat{n}_b(k) - S(k) + \hat{\mu}(k)RTT(k) + \hat{\mu}(k+1)\hat{RTT}(k+1)] \qquad 5.8$$

Since both $\hat{\mu}(k)$ and $\hat{\mu}(k+1)$ are unknown, we can safely assume that $\hat{\mu}(k) = \hat{\mu}(k+1)$. Further, from an earlier assumption, we set $\widehat{RTT}(k+1)$ to $RTT(k)$. This gives us:

$$\lambda(k+1) = \frac{1}{RTT(k)}[B/2 - \hat{n}_b(k) - S(k) + 2\hat{\mu}(k)RTT(k)] \qquad 5.9$$

This is the control law. The control always tries to obtain a queue length in the bottleneck equal to $B/2$. It may never reach there, but will always stay around it.

Note that the control law requires us to maintain two estimators: $\hat{\mu}(k)$ and $\hat{n}_b(k)$. The effectiveness of the control depends on the choice of the estimators. This is considered in sections 5 and 6.

5.3.5. Stability analysis

The state equation is given by (5.2)

$$n_b(k+1) = n_b(k) + \lambda(k)RTT(k) - \mu(k)RTT(k) \qquad 5.10$$

For the stability analysis of the controlled system, $\lambda(k)$ should be substituted using the control law. Since we know $\lambda(k+1)$, we use the state equation derived from (5.2) instead (which is just one step forward in time). This gives

$$n_b(k+2) = n_b(k+1) + (\lambda(k+1)-\mu(k+1))RTT(k+1)$$

Substituting (5.8) in (5.10), we find the state evolution of the controlled system:

$$n_b(k+2) = n_b(k+1) - \mu(k+1)RTT(k+1) + \frac{RTT(k+1)}{RTT(k)}$$
$$[B/2 - \hat{n}_b(k) - S(k) + 2\hat{\mu}(k)RTT(k)]$$

By assumption, $RTT(k)$ is close to $RTT(k+1)$. So, to first approximation, canceling $RTT(k)$ with $RTT(k+1)$ and moving back two steps in time,

$$n_b(k) = n_b(k-1) - \mu(k-1)RTT(k-1) +$$
$$B/2 - \hat{n}_b(k-2) - S(k-2) + 2\hat{\mu}(k-2)RTT(k-2)$$

Taking the Z transform of both sizes, and assuming $n_b(k-2) = \hat{n}_b(k-2)$, we get

$$n_b(z) = z^{-1}n_b(z) - z^{-2}\mu(z)*RTT(z) +$$
$$B/2 - z^{-2}n_b(z) - z^{-2}S(z) + 2z^{-4}\hat{\mu}(z)*RTT(z)$$

Considering n_b as the state variable, it can be easily shown that the characteristic equation is

$$z^{-2} - z^{-1} + 1 = 0$$

If the system is to be asymptotically stable, then the roots of the characteristic equation (the eigenvalues of the system), must lie inside the unit circle on the complex Z plane. Solving for z^{-1}, we get

$$z^{-1} = \frac{1 \pm \sqrt{1-4}}{2} = \frac{1 \pm i\sqrt{3}}{2}$$

The distance from 0 is hence

$$\sqrt{\frac{1}{2}^2 + \frac{\sqrt{3}}{2}^2} = 1$$

Since the eigenvalues lie on the unit circle, the controlled system is *not* asymptotically stable.

However, we can place the pole of the characteristic equation so that the system is asymptotically stable. Consider the control law

$$\lambda(k+1) = \frac{\alpha}{RTT(k)}$$
$$[B/2 - \hat{n}_b(k) - S(k) + 2\hat{\mu}(k)RTT(k)]$$

This leads to a characteristic equation

$$\alpha z^{-2} - z^{-1} + 1 = 0$$

so that the roots are

$$z^{-1} = \frac{1 \pm i\sqrt{4\alpha - 1}}{2\alpha}$$

The poles are symmetric about the real axis, so we need only ensure that

$$|z^{-1}| > 1$$

$$=> \sqrt{(\frac{1}{2\alpha})^2 + (\frac{\sqrt{4\alpha - 1}}{2\alpha})^2} > 1$$

$$=> \frac{1}{\sqrt{\alpha}} > 1 \; => \; \alpha < 1$$

This means that if $\alpha < 1$, the system is provably asymptotically stable (by the Separation Theorem, since the system and observer eigenvalues are distinct, this stability result holds irrespective of the choice of the estimators).

The physical interpretation of α is simple: to reach $B/2$ at the end of the next epoch, the source should send exactly at the rate computed by (9). If it does so, the system may be unstable. Instead, it sends at a slightly lower rate, and this ensures that the system is asymptotically stable. Note that α is a constant that is independent of the system's dynamics and can be chosen in advance to be any desired value smaller than 1.0. The exact value chosen for α controls the rise time of the system, and, for adequate responsiveness, it should not be too small. Our simulations indicate that a value of 0.9 is a good compromise between responsiveness and instability. Similar studies are mentioned in [30].

5.4. System non-linearity

This section discusses a non-linearity in the system, and how it can be accounted for in the analysis. The state equation (5.9) is correct when $n_b(k+1)$ lies in the range 0 to B. Since the system is physically incapable of having less than zero and more than B packets in the bottleneck queue, the equation actually is incorrect at the endpoints of this range. The correct equation is then:

$$n_b(k+1) = \begin{bmatrix} \text{if } n_b(k) + S(k) - RTT(k)\mu(k) < 0 \textit{ then } 0 \\ \text{if } n_b(k) + S(k) - RTT(k)\mu(k) > B \textit{ then } B \\ \textit{otherwise} \;\; n_b(k) + S(k) - RTT(k)\mu(k) \end{bmatrix}$$

The introduction of the inequalities in the state equation makes the system nonlinear at the boundaries. This is a difficulty, since the earlier proof of stability is valid only for a linear system. However, note that, if the equilibrium point (setpoint) is chosen to lie inside the range [0,B], then the system is linear around the setpoint. Hence, for small deviations from the setpoint, the earlier stability proof, which assumes linearity, is sufficient. For large deviations, stability must be proved by other methods, such as the second method of Liapunov ([102] page 558).

However, this is only an academic exercise. In practice, the instability of the system means that n_b can move arbitrarily away from the setpoint. In section 10.2, we show how window-based flow control can be used in conjunction with a rate-based approach. Then, since n_b can never be less than 0, and the window flow control protocol ensures that it never exceeds B, true instability is not possible.

Nevertheless, we would like the system to return to the setpoint, whenever it detects that it has moved away from it, rather than operating at an endpoint of its range. This is automatically assured by equation (9), which shows that the system chooses $\lambda(k+1)$ such that $n_b(k+2)$ is $B/2$. So, whenever the system detects that it is at an endpoint, it immediately takes steps to ensure that it moves away from it.

Thus, the non-linearity in the system is of no practical consequence, except that the flow control mechanism has to suitably modify the state equations when updating $\hat{n}_b(k+1)$. A rigorous proof of the stability of the system using Liapunov's second method is also possible, but the gain from the analysis is slight.

5.5. Kalman state estimation

A practical scheme is presented in §5.6. Having derived the control law, and proved its stability, we now need to determine stable estimators for the system state. §5.5 presents a Kalman state estimator, and shows that Kalman estimation is impractical. We choose to use Kalman estimation, since it is a well known and robust technique [49]. Before the technique is applied, a state-space description of the system is necessary.

5.5.1. State space description

We will use the standard linear stochastic state equation given by

$$\mathbf{x}(k+1) = \mathbf{Gx}(k) + \mathbf{Hu}(k) + \nu_1(k)$$
$$\mathbf{y}(k) = \mathbf{Cx}(k) + \nu_2(k)$$

x, **u** and **y** are the state, input and output vectors of sizes n, m, and r, respectively. **G** is the nxn state matrix, **H** is an nxm matrix, and **C** is an rxn matrix. ν_1(k) represents the system noise vector, which is assumed to be zero-mean, gaussian and white. $\nu_2(k)$ is the observation noise, and it is assumed to have the same characteristics as the system noise.

Clearly, **u** is actually u, a scalar, and $u(k) = \lambda(k)$. At the end of epoch k, the source receives probes from epoch k-1. (To be precise, probes can be received from epoch k-1 as well as from the beginning of epoch k. However, without loss of generality, this is modeled as part of the observation noise.) So, at that time, the source knows the average service time in the k-1th epoch, $\mu(k-1)$. This is the only observation it has about the system state, and so $y(k)$ is a scalar, $y(k) = \mu(k-1) + \nu_2$. If $y(k)$ is to be derived from the state vector **x** by multiplication with a constant matrix, then the state must contain $\mu(k-1)$. Further, the state must also include the number of packets in the bottleneck's buffer, n_b. This leads to a state vector that has three elements, n_b, $\mu(k)$, and $\mu(k-1)$, where $\mu(k)$ is needed since it is part of the delay chain leading to $\mu(k-1)$ in the corresponding signal flow graph. Thus,

$$\mathbf{x} = \begin{bmatrix} n_b \\ \mu \\ \mu_{-1} \end{bmatrix}$$

where μ_{-1} represents the state element that stores the one-step delayed value of μ.

We now turn to the **G**, **H**, ν_1, ν_2 and **C** matrices. The state equations are

$$n_b(k+1) = n_b(k) + \lambda(k)RTT(k) - \mu(k)RTT(k)$$

$$\mu(k+1) = \mu(k) + \omega(k)$$

$$\mu_{-1}(k+1) = \mu(k)$$

Since $RTT(k)$ is known at the end of the kth epoch, we can represent it by a pseudo-constant, Rtt. This gives us the matrices

$$\mathbf{G} = \begin{bmatrix} 1 -Rtt\ 0 \\ 0\ 1\ 0 \\ 0\ 1\ 0 \end{bmatrix}$$

$$\mathbf{H} = \begin{bmatrix} Rtt \\ 0 \\ 0 \end{bmatrix}$$

$$\nu_1 = \begin{bmatrix} 0 \\ \omega \\ 0 \end{bmatrix}$$

$$\mathbf{C} = [\ 0\ 0\ 1\ ]$$

ν_2 is simply the (scalar) variance in the observation noise. This completes the state space description of the flow control system.

5.5.2. Kalman filter solution to the estimation problem

A Kalman filter is the minimum variance state estimator of a linear system. In other words, of all the possible estimators for **x**, the Kalman estimator is the one that will minimize the value of $E([\hat{x}(t) - x(t)]^T[\hat{x}(t) - x(t)])$, and in fact this value is zero. Moreover, a Kalman filter can be manipulated to yield many other types of filters [49]. Thus, it is desirable to construct a Kalman filter for **x**.

In order to construct the filter, we need to determine three matrices, **Q, S** and **R**, which are defined implicitly by :

$$E\left\{\begin{bmatrix}\nu_1(k)\\ \nu_2(k)\end{bmatrix}[\nu_1^T(\theta)\nu_2(\theta)]\right\} = \begin{bmatrix}Q & S\\ S^T & R\end{bmatrix}\delta\,(t - \theta)$$

(where δ is the Kronecker delta defined by, $\delta(k) =$ if (k = 0) then 1 else 0). Expanding the left hand side, we have

$$\mathbf{Q} = \mathrm{E}\begin{bmatrix}0 & 0 & 0\\ 0 & \omega^2 & 0\\ 0 & 0 & 0\end{bmatrix}$$

$$\mathbf{R} = \mathrm{E}\ (\nu_2{}^2)$$

$$\mathbf{S} = \mathrm{E}\begin{bmatrix}0\\ \omega\nu_2\\ 0\end{bmatrix}$$

If the two noise variables are assumed to be independent, then the expected value of their product will be zero, so that $\mathbf{S} = \mathbf{0}$. However, we still need to know $\mathrm{E}(\omega^2)$ and $\mathrm{E}(\nu_2{}^2)$.

From the state equation,

$$\mu(k+1) = \mu(k) + \omega(k)$$

Also,

$$\mu_{observed}(k+1) = \mu(k+1) + \nu_2(k+1)$$

Combining,

$$\mu_{observed}(k+1) = \mu(k) + \omega(k) + \nu_2(k+1)$$

which indicates that the observed value of μ is affected by both the state and observation noise. As such, each component cannot be separately determined from the observations alone. Thus, in order to do Kalman filtering, the values of $\mathrm{E}(\omega^2)$ and $\mathrm{E}(\nu_2{}^2)$ must be extraneously supplied, either by simulation or by measurement of the actual system. Practically speaking, even if good guesses for these two values are supplied, the filter will have reasonable (but not optimal) performance. Hence, we will assume that the values of the noise variances are supplied by the system administrator, and so matrices **Q, R** and **S** are known. It is now straightforward to apply Kalman filtering to the resultant system. We follow the derivation in [49] (pg 249).

The state estimator $\hat{x}$ is derived using

$$\hat{\mathbf{x}}(k+1) = \mathbf{G}\hat{\mathbf{x}}(k) + \mathbf{K}(k)[y(k) - \mathbf{C}\hat{\mathbf{x}}(k)] + \mathbf{H}u(k)$$

$$\hat{\mathbf{x}}(0) = \mathbf{0}$$

where **K** is the Kalman filter gain matrix, and is given by

$$\mathbf{K}(k) = [\mathbf{G}\Sigma(k)\mathbf{C}^T + \ \mathbf{S}\ ][\mathbf{C}\Sigma(k)\mathbf{C}^T + \mathbf{R}]^{-1}$$

$\Sigma(k)$ is the error state covariance, and is given by the Riccati difference equation

$$\Sigma(k+1) = \mathbf{G}\Sigma(k)\mathbf{G}^T + \mathbf{Q} - \mathbf{K}(k)[\mathbf{C}\Sigma(k)\mathbf{C}^T + \mathbf{R}]\mathbf{K}(k)^T$$

$$\Sigma(0) = \Sigma_0$$

where Σ_0 is the covariance of **x** at time 0, and can be assumed to be **0**.

Note that a Kalman filter requires the Kalman gain matrix **K**(k) to be updated at each time step. This computation involves a matrix inversion, and appears to be generally expensive. However, since all the matrices are at most 3x3, in practice this is not a problem.

To summarize, if the variances of the system and observation noise are available, Kalman filtering is an attractive estimation technique. However, if these variances are not available, then Kalman filtering cannot be used. In the next section, we present a heuristic estimator that works even in the absence of knowledge about system and observation noise.

5.6. Fuzzy estimation

This section presents the design of a fuzzy system that predicts the next value of a time series. Consider a scalar variable θ that assumes the sequence of values

$$\{\theta_k\} = \theta_{1,}\ \theta_2,\ \cdots,\ \theta_k$$

where

$$\theta_k = \theta_{k-1} + \omega_{k-1}$$

and ω_k (called the ‘system perturbation’) is a random variable from some unknown distribution.

Suppose that an observer sees a sequence of values

$$\tilde{\theta}_{1,}\ \tilde{\theta}_{2,}\ .....,\ \tilde{\theta}_{k-1}$$

and wishes to use the sequence to estimate the current value of θ_k. We assume that the observed sequence is corrupted by some observation noise ξ, so that the observed values $\{\tilde{\theta}_k\}$ are not the actual values $\{\theta_k\}$, and

$$\tilde{\theta}_k = \theta_k + \xi_k$$

where ξ_k is another random variable from an unknown distribution.

Since the perturbation and noise variables can be stochastic, the exact value of θ_k cannot be determined. What is desired, instead, is $\hat{\theta}_k$, the predictor of θ_k, be optimal in some sense.

5.6.1. Assumptions

We model the parameter θ_k as the state variable of an unknown dynamical system. The sequence $\{\theta_k\}$ is then the sequence of states that the system assumes. We make three weak assumptions about the system dynamics. First, the time scale over which the system perturbations occur is assumed to be an order of magnitude slower than the corresponding time scale of the observation noise.

Second, we assume that system can span a spectrum ranging from ‘steady’ to ‘noisy’. When it is steady, then the variance of the system perturbations is close to zero, and changes in { $\tilde{\theta}_k$ } are due to observation noise. When the system is noisy, $\{\theta_k\}$ changes, but with a time constant that is longer than the time constant of the observation noise. Finally, we assume that ξ is from a zero mean distribution.

Note that this approach is very general, since there are no assumptions about the exact distributions of ω and ξ. On the other hand, there is no guarantee that the resulting predictor is optimal: we only claim that the method is found to work well in practice.

5.6.2. Exponential averaging

The basis of this approach is the predictor given by:

$$\hat{\theta}_{k+1} = \alpha\hat{\theta}_k + (1-\alpha)\tilde{\theta}_k$$

The predictor is controlled by a parameter α, where α is the weight given to past history. The larger it is, the more weight past history has in relation to the last observation. The method is also called exponential averaging, since the predictor is the discrete convolution of the observed sequence with an exponential curve with a time constant α:

$$\hat{\theta}_k = \sum_{i=0}^{k-1}(1-\alpha)\tilde{\theta}_i\alpha^{k-i-1} + \alpha^k\hat{\theta}_0$$

The exponential averaging technique is robust, and so it has been used in a number of applications. However, a major problem with the exponential averaging predictor is in the choice of α. While in principle, it can be determined by knowledge of the system and observation noise variances, in practice, these variances are unknown. It would be useful to automatically determine a 'good' value of α, and to be able to change this value on-line if the system behavior changes. Our approach uses fuzzy control to effect this tuning [82, 152, 158].

5.6.3. Fuzzy exponential averaging

Fuzzy exponential averaging is based on the heuristic that a system can be thought of as belonging to a spectrum of behaviors that ranges from 'steady' to 'noisy'. In a 'steady' system ($\omega \ll \xi$), the sequence $\{\theta_k\}$ is approximately constant, so that $\{\tilde{\theta}_k\}$ is affected mainly by observation noise. Then, α should be large, so that the past history is given more weight, and transient changes in $\tilde{\theta}$ are ignored.

In contrast, if the system is 'noisy' ($\omega \approx \xi$ or $\omega > \xi$), $\{\theta_k\}$ itself could vary considerably, and $\tilde{\theta}$ reflects changes both in θ_k and the observation noise. By choosing a lower value of α, the observer quickly tracks changes in θ_k, while ignoring past history which only provides old information.

While the choice of α in the extremal cases is simple, the choice for intermediate values along the spectrum is hard to make. We use a fuzzy controller to determine a value of α that gracefully responds to changes in system behavior. Thus, if the system moves along the noise spectrum, α adapts to the change, allowing us to obtain a good estimate of θ_k at all times. Moreover, if the observer does not know α *a priori*, the predictor automatically determines an appropriate value.

5.6.4. System identification

Since α is linked to the 'noise' in the system, how can the amount of 'noise' in the system be determined? Assume, for the moment, that the variance in ω is an order of magnitude larger than the variance in ξ. Given this assumption, if a system is 'steady', the exponential averaging predictor will usually be accurate, and prediction errors will be small. In this situation, α should be large. In contrast, if the system is 'noisy', then the exponential averaging predictor will have a large estimation error. This is because, when the system noise is large, past history cannot predict the future. So, no matter the value of α, it will usually have a large error. In that case, it is best to give little weight to past history by choosing a small value of α, so that the observer can track the changes in the system.

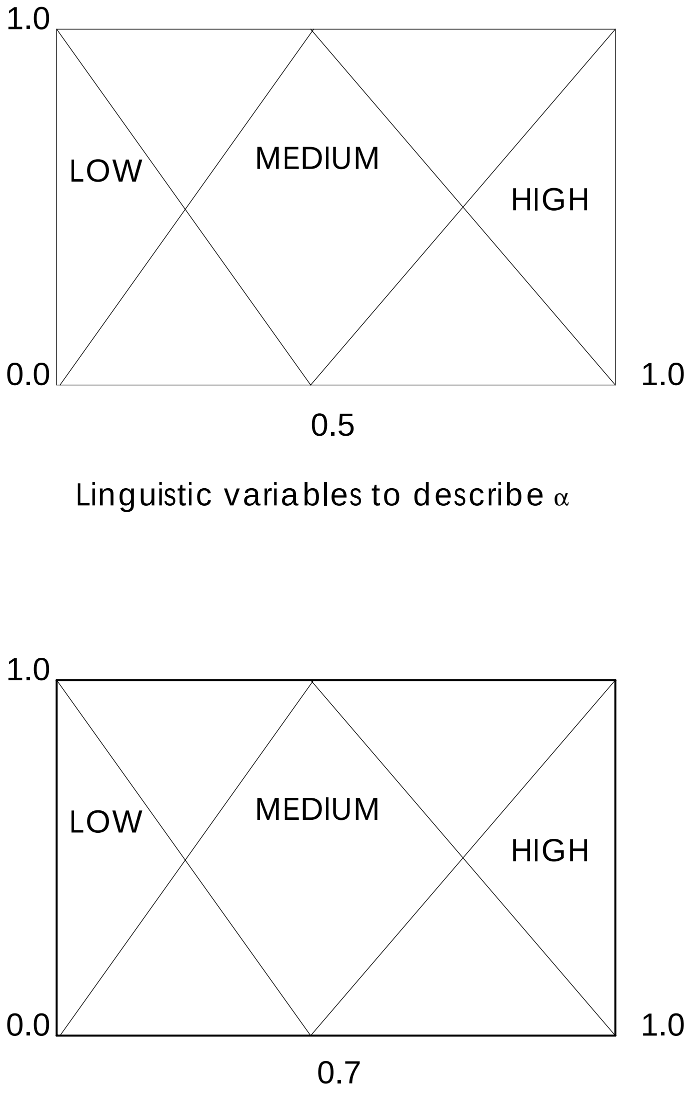
To summarize, we have observed that, if the predictor error is large, then α should be small, and vice versa. Treating 'small' and 'large' as fuzzy linguistic variables [151], we can build a fuzzy controller for the estimation of α.

5.6.5. Fuzzy controller

The controller implements three fuzzy laws:

If error is low, then α is high
If error is medium, then α is medium
If error is high, then α is low

The linguistic variables ‘low’, ‘medium’ and ‘high’ for α and error are defined in Figure 5.2.



Linguistic variables to describe α

Linguistic variables to describe the error

Figure 5.2: Definition of linguistic variables

The input to the fuzzy controller is a value of the error, and the controller outputs α in three steps. First, the error value is mapped to a membership in each of the fuzzy sets ‘low’, ‘medium’ and ‘high’ using the definition in Figure 5.3. Then, the control rules are used to determine the applicability of each outcome to the resultant control. Finally, the fuzzy set representing the control is defuzzified using a centroid defuzzifier.

The error $|\tilde{\theta} - \hat{\theta}|$ is processed in two steps before it is input to the fuzzy system. First, it is converted to a relative value given by error $= \frac{|\tilde{\theta}_k - \hat{\theta}_k|}{\tilde{\theta}_k}$. It is not a good idea to use the relative error value directly, since spikes in $\tilde{\theta}_k$ can cause the error to be large, α would drop to 0, and all past history would be lost. So, in the second step, the relative error is smoothed using another exponential averager. The constant for this averager, β, is obtained from another fuzzy controller that links the change in the error to the value of β. The idea is that, if the change in error is large, then β should be large, so that spikes are ignored. Otherwise, β should be small. β and the change in error are defined by the same linguistic variables, 'low', 'medium' and 'high', and these are defined exactly like the corresponding variables for α. With these changes, the assumption that the variance in the observation noise is small can now be removed. The resulting system is shown in Figure 5.3.

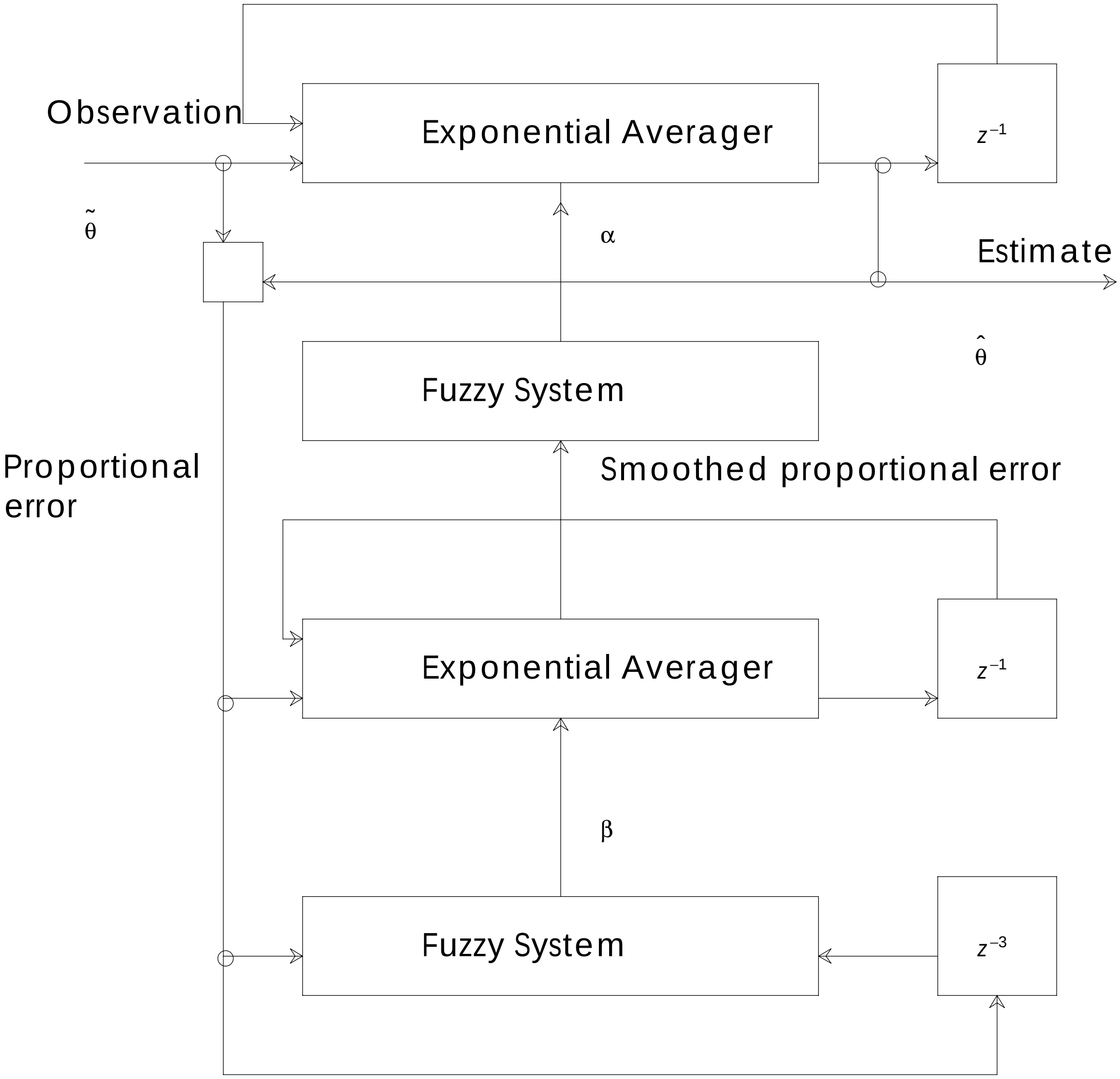


Figure 5.3: Fuzzy prediction system

Details of the prediction system, and a performance analysis can be found in reference [75].

5.7. Using additional information

This section describes how the frequency of control can be increased by using information about the propagation delay. Note that $\hat{n}_b(k+1)$, the estimate for the number of packets in the bottleneck queue, plays a critical role in the control system. The controller tracks changes in $\hat{n}_b(k)$, and so it is necessary that $\hat{n}_b(k)$ be a good estimator of n_b. $\hat{n}_b(k)$ can be made more accurate if additional information from the network is available. One such piece of information is the value of the propagation delay.

The round-trip time of a packet includes delays due to three causes:

- the propagation delay due to the finiteness of the speed of light and the processing at switches and interfaces
- the queueing delay at each switch, because previous packets from that conversation have not yet been serviced
- the phase delay, introduced when the first packet from a previously inactive conversation waits for the server to finish service of packets from other conversations

The propagation delay depends on the geographical spread of the network, and for WANs, it can be of the order of a few tens of milliseconds. The phase delay is roughly the same magnitude as the time it takes to process one packet each from all the conversations sharing a server, the *round time*. The queueing delay is of the order of several round times, since each packet in the queue takes one round time to get service. For future high speed networks, we expect the propagation and queueing delays to be of roughly the same magnitude, and the phase delay to be one order of magnitude smaller. Thus, if queueing delays can be avoided by the probe packet, the measured round-trip time will be approximately the propagation delay of the conversation.

An easy way to avoid queueing delays is to measure the round-trip time for the first packet of the first packet-pair. Since this packet has no queueing delays, we can estimate the propagation delay of the conversation from this packet's measured round trip time (though it has a component due to phase delay). Call this propagation delay R.

The value of R is useful, since the number of packets in the bottleneck queue at the beginning of epoch $k+1$, $n_b(k+1)$, can be estimated by the number of packets being transmitted ('in the pipeline') subtracted from the number of unacknowledged packets at the beginning of the epoch, $S(k)$. That is,

$$\hat{n}_b(k+1) = S(k) - R\hat{\mu}(k)$$

Since S, R and $\hat{\mu}(k)$ are known, this gives us another way of determining $\hat{n}_b(k+1)$. This can be used to update $\hat{n}_b(k+1)$ as an alternative to equation (2). The advantage of this approach is that equation (2) is more susceptible to parameter drift. That is, successive errors in $\hat{n}_b(k+1)$ can add up, so that $\hat{n}_b(k+1)$ could differ substantially from n_b. In the new scheme, this risk is considerably reduced: the only systematic error that could be made is in μ, and since this is frequent sampled, as well as smoothed by the fuzzy system, this is of smaller concern.

There is another substantial advantage to this approach: it enables control actions to be taken much faster than once per round trip time. This is explained in the following section.

5.7.1. Faster than once per RTT control

It is useful to take control actions as fast as possible so that the controller can react immediately to changes in the system. In the system described thus far, we limited ourselves to once per RTT control because this allows us to use the simple relationship between $S(k)$ and $\lambda(k)$ given by equation (1). If control actions are taken faster than once per RTT, then the epoch size is smaller, and that relationship is no longer true. The new relationship is much more complicated, and it is easily shown that the state and input vectors must expand to include time delayed values of μ, λ and n_b. It is clear that the faster the control actions are required, the larger the state vector, and this complicates both the analysis and the control.

In contrast, with information about the propagation delay R, control can be done as quickly as once every packet-pair with no change to the length of the state vector. This is demonstrated below.

We will work in continuous time, since this makes the analysis easier. We also make the fluid approximation [1], so packet boundaries are ignored, and the data flow is like that of a fluid in a hydraulic system. This approximation is commonly used [8, 133], and both analysis [97] and simulations show that the approximation is a close one.

Let us assume that λ, the sending rate, is held fixed for some duration J, starting from time t. Then,

$$n_b(t+J) = n_b(t) + \lambda(t)J - \mu(t)J \qquad 5.11$$

where μ is the average service rate in the time interval $[t, t+J]$, and n_b is assumed to lie in the linear region of the space. Also, note that

$$n_b(t) = S(t) - R\mu(t) \qquad 5.12$$

The control goal is to have $n_b(t+J)$ be the setpoint value $B/2$. Hence,

$$n_b(t+J) = n_b(t) + \lambda(t)J - \mu(t)J = B/2 \qquad 5.13$$

So,

$$\lambda(t) = \frac{B/2 - S(t) + R\hat{\mu}(t) + J\hat{\mu}(t)}{J} \qquad 5.14$$

which is the control law. The stability of the system is easily determined. Note that $\dot{n}_b(t)$ is given by

$$\dot{n}_b(t) = \underset{\delta \to 0}{limit} \frac{n_b(t+\delta) - n_b(t)}{\delta} = \lambda(t) - \mu(t) \qquad 5.15$$

From equation (5.13),

$$\dot{n}_b = \frac{B/2 - n_b(t)}{J} \qquad 5.16$$

If we define the state of the system by

$$x = n_b(t) - B/2 \qquad 5.17$$

then the equilibrium point is given by

$$x = 0 \qquad 5.18$$

and the state equation is

$$\dot{x} = \frac{-x}{J} \qquad 5.19$$

Clearly, the eigenvalue of the system is -1/J, and since J is positive, the system is both Lyapunov stable and asymptotically stable. In this system, J is the pole placement parameter, and plays exactly the same role as α in the discrete time system. When J is close 0, the eigenvalue of the system is close to $-\infty$ and the system will reach the equilibrium point rapidly. Larger values of J will cause the system to move to the equilibrium point more slowly. An intuitively satisfying choice of J is one round trip time, and this is easily estimated as $R + S(k)\mu(t)$. In practice, the values of R and $S(k)$ are known, and $\mu(t)$ is estimated by $\hat{\mu}$, which is the fuzzy predictor described earlier.

5.8. Practical issues

This section considers two practical problems: how to correct for parameter drift; and how to coordinate rate-based and window-based flow control.

5.8.1. Correcting for parameter drift

In any system with estimated parameters, there is a possibility that the estimators will drift away from the true value, and that this will not be detected. In our case, the estimate for the number of packets in the bottleneck buffer at time k, $\hat{n}_b(k)$, is computed from $\hat{n}_b(k-1)$ and from the estimator $\hat{\mu}(k)$. If the estimators are incorrect, $\hat{n}_b(k)$ might drift away from $n_b(k)$. Hence, it is reasonable to require a correction for parameter drift.

Note that, if $\lambda(k)$ is set to 0 for some amount of time, then n_b will decrease to 0. At this point, $\hat{n}_b$ can also be set to 0, and the system will resynchronize. In practice, the source sends a special pair and then sends no packets till the special pair is acknowledged. Since no data was sent after the pair, when acks are received, the source is sure that the bottleneck queue has gone to 0. It can now reset $\hat{n}_b$ and continue.

The penalty for implementing this correction is the loss of bandwidth for one round trip time. If a conversation lasts over many round trip times, then this loss may be insignificant over the lifetime of the conversation. Alternately, if a user sends data in bursts, and the conversation is idle between bursts, then the value of $\hat{n}_b$ can be resynchronized to 0 one RTT after the end of the transmission of a data burst.

5.8.2. The role of windows

Note that our control system does not give us any guarantees about the shape of the buffer size distribution $N(x)$. Hence, there is a non-zero probability of packet loss. In many applications, packet loss is undesirable. It requires endpoints to retransmit messages, and frequent retransmissions can lead to congestion. Thus, it is desirable to place a sharp cut-off on the right end of $N(x)$, or, strictly speaking, to ensure that there are no packet arrivals when $n_b = B$. This can be arranged by having a window flow control algorithm operating simultaneously with the rate-based flow control algorithm described here.

In this scheme, the rate-based flow control provides us a ‘good’ operating point which is the setpoint that the user selects. In addition, the source has a limit on the number of packets it could have outstanding (the window size), and every server on its path reserves at least a window’s worth of buffers for that conversation. This assures us that, even if the system deviates from the setpoint, the system does not lose packets and possible congestive losses are completely avoided.

Note that, by reserving buffers per conversation, we have introduced reservations into a network that we earlier claimed to be reservationless. However, our argument is that strict *bandwidth* reservation leads to a loss of statistical multiplexing. As long as no conversation is refused admission due to a lack of buffers, statistical multiplexing of bandwidth is not affected by buffer reservation, and the multiplexing gain is identical to that received in a network with no buffer reservations. Thus, with large cheap memories, we claim that it will be always be possible to reserve enough buffers so that there is no loss of statistical multiplexing.

To repeat, we use rate-based flow control to select an operating point, and window-based flow control as a conservative cut-off point. In this respect, we agree with Jain that the two forms of flow control are *not* diametrically opposed, but in fact can work together [67].

The choice of window size is critical. Using fixed size windows is usually not possible in high speed networks, where the bandwidth-delay product, and hence the required window, can be large (of the order of hundreds of kilobytes per conversation). In view of this, the adaptive window allocation scheme proposed by Hahne *et al* [54] is attractive. In that scheme, a conversation is allocated a flow control window that is always larger than the product of the allocated bandwidth at the bottleneck, and the round trip propagation delay. So, a conversation is never constrained by the size of the flow control window. A signaling scheme dynamically adjusts the window size in response to changes in the network state. We believe that their window-based flow control scheme is complementary to the rate-based flow control scheme proposed in this chapter.

5.9. Limitations of the control-theoretic approach

The main limitation of a control-theoretic approach is that it restricts the form of the system model. Since most control-theoretic results hold for linear systems, the system model must be cast in this form. This can be rather restrictive, and certain aspects of the system, such as the window flow control scheme, are not adequately modeled. Similarly, the standard noise assumptions are also restrictive, and may not reflect the actual noise distribution in the target system. These are mainly the limitations of linear control. There is a growing body of literature dealing with non-linear control, and one direction for future work would be to study non-linear models for flow control.

Another limitation of control theory is that, for controller design, the network state be observable. Since a FCFS server's state cannot be easily observed, it is hard to apply control theoretic principles to the control of FCFS networks. In contrast, FQ state can be probed using a packet pair, and so FQ networks are amenable to a formal treatment.

5.10. Related work and contributions

Several control-theoretic approaches to flow control have been studied in the past. One body of work has considered the dynamics of a system where users update their sending rate either synchronously or asynchronously in response to measured round trip delays, or explicit congestion signals, for example in references [6,9,10,27,127]. These approaches typically assume Poisson sources, availability of global information, a simple flow update rule, and exponential servers. We do not make such assumptions. Further, they deal with the dynamics of the entire system, and take into account the sending rate of all the users explicitly. In contrast, we consider a system with a single user, where the effects of the other users are considered as a system 'noise'. Also, in our approach, each user uses a rather complex flow update rule, based in part on fuzzy prediction, and so the analysis is not amenable to the simplistic approach of these authors.

Some control principles have been appealed to in work by Jain [116] and Jacobson [63], but the approaches of these authors are quite informal. Further, their control systems take multiple round trip times to react to a change in the system state. In contrast, the system in §5.9.1 can take control action multiple times per RTT. In a high bandwidth-delay product network, this is a significant advantage.

In recent work, Ko *et al* [79] have studied an almost identical problem, and have applied principles of predictive control to hop-by-hop flow control. However, they appeal primarily to intuitive heuristics, and do not use a formal control-theoretic model; hence they are not able to prove the stability of their system. Further, we believe that our fuzzy scheme is a better way to predict service rates than their straightforward moving-average approach.

A control-theoretic approach to individual optimal flow control was described originally by Agnew [1], and since extended by Filipiak [38] and Tipper *et al* [133]. In their approach, a conversation is modeled by a first order differential equation, using the fluid approximation. The modeling parameters are tuned so that, in the steady state, the solution of the differential equation and the solution of a corresponding queueing model agree. While we model the service rate at the bottleneck μ as a random walk, they assume that the service rate is a non-linear function of the global queue length (over all conversations), so that $\mu = G(n_b)$, where $G(.)$ is some non-linear function. This is not true for a FQ server, where the service rate is independent of the queue length. Hence, we cannot apply their techniques to our problem.

Vakil, Hsiao and Lazar [137] have used a control-theoretic approach to optimal flow control in double-bus TDMA local-area integrated voice/data networks. However, they assume exponential FCFS servers, and, since the network is not geographically dispersed, propagation delays are ignored. Their modeling of the service rate μ is as a random variable instead of a random walk, and, though they propose the use of recursive minimum mean squared error filters to estimate system state, the bulk of the results assume complete information about the network state. Vakil and Lazar [138] have considered the design of optimal traffic filters when the state is not fully observable, but the filters are specialized for voice traffic.

Robertazzi and Lazar [119] and Hsiao and Lazar [60] have shown that, under a variety of conditions, the optimal flow control for a Jacksonian network with Poisson traffic is *bang-bang* (approximated by a window scheme). It is not clear that this result holds when their strong assumptions are removed.

In summary, we feel that our approach is substantially different from those in the literature. Our use of a packet pair to estimate the system state is unique, and this estimation is critical in enabling the control scheme. We have described two provably stable rate-based flow control schemes as well as a novel estimation scheme using fuzzy logic. Some practical concerns in implementing the scheme have also been addressed.

The control law presented in §5.9.1 has been extensively simulated in a number of scenarios and the results are presented in Chapter 6. The results can be summarized as

- The performance of the flow control with Fair Queueing servers in the benchmark suite described in reference [23] is comparable to that of the DECbit scheme [117], but without any need for switches to set bits.
- The flow control algorithm responds quickly and cleanly to changes in network state.
- Unlike some current flow control algorithms (DECbit and Jacobson’s modifications to 4.3 BSD TCP [63, 117]), the system behaves extraordinarily well in situations where the bandwidth-delay product is large, even if the cross traffic is misbehaved or bursty.
- Implementation and tuning of the algorithm is straightforward, unlike the complex and ad-hoc controls in current flow control algorithms.
- Even in complicated scenarios, the dynamics are simple to understand and manage: in contrast the dynamics of Jacobson’s algorithm are messy and only partially understood [156].

In conclusion, we believe that our decision to use a formal control-theoretic approach in the design of a flow control algorithm has been a success. Our algorithm behaves well even under great stress, and, more importantly, it is simple to implement and tune. These are not fortuitous, rather, they reflect the theoretical underpinnings of the approach.

5.11. Future work

This chapter makes several simplifications and assumptions. It would be useful to measure real networks to see how far theory and practice agree. We plan to make such measurements in the XUNET II experimental high speed network testbed [69]. Other possible extensions are to design a minimum variance controller and a non-linear controller.

5.12. Appendix 5.A - Steady state

As a sanity check, consider the steady state where μ does not change. We prove that in this case λ is set to μ, and that the number of packets in the bottleneck will be $B/2$.

In the steady state, any sensible estimator $\hat{\mu}(k)$ will converge, so that $\hat{\mu}(k) = \hat{\mu}(k+1) = \mu$. We will assume that we are starting at time 0, that $n_b(0) = 0$ and that $\alpha = 1$.

In the steady state, the state equation (2) becomes

$$n_b(k+1) = n_b(k) + \lambda(k+1)RTT(k+1) - \mu(k+1)RTT(k+1)$$

We assume that the estimators converge to the correct value, since there is no stochastic variation in the system. Hence,

$$\hat{RTT} = RTT \text{ for } \textit{all } k$$

This makes the control law

$$\lambda(k+1) = \frac{1}{RTT(k)}[B/2 - \hat{n}_b(k) - S(k) + \mu RTT(k) + \mu RTT(k)]$$

$$\lambda(k+1) = 2\mu - \lambda(k) + \frac{1}{RTT(k)}[B/2 - \hat{n}_b(k)]$$

The first packet pair estimates μ, and, for simplicity, we assume that this is exactly correct. Hence,

$$\lambda(1) = 2\mu - \mu + \frac{1}{RTT(1)}(B/2)$$

$$\lambda(1) = \mu + \frac{B/2}{RTT(1)}$$

The buffer at the end of epoch 1 is given by

$$n_b(1) = 0 + (\mu + \frac{B/2}{RTT(1)})RTT(1) - \mu RTT(1)$$

$$n_b(1) = B/2$$

To check further, at time 2, we get

$$\lambda(2) = 2\mu - \mu + \frac{1}{RTT(2)}[B/2 - B/2]$$

and, since $\hat{n}_b(k+1)(1) = B/2$.

$$=> \lambda(2) = \mu$$

And,

$$n_b(2) = B/2 + \mu RTT(2) - \lambda(2)RTT(2)$$

$$n_b(2) = B/2 \qquad 5.A1$$

So, the buffer is B/2 at time 2. Let us see what λ(3) is when λ(2) is μ.

$$\lambda(3) = 2\mu - \mu + \frac{1}{RTT(3)}[B/2–B/2]$$

$$\lambda(3) = \mu$$

So, if $\lambda(k) = \mu$ and $n_b(k) = B/2$, $\lambda(k+1) = \mu$, and λ is fixed from time 3 onwards. From (5.A1) we see that if $\lambda = \mu$ and $n_b(k) = B/2$ then $n_b(k+1) = B/2$. Thus, both the recurrences reach the stable point at $(\mu, B/2)$ at time 3. Thus, if the system is steady, so is the control. This is reassuring.

5.13. Appendix 5.B - Linear quadratic gaussian optimal control

This section considers an approach to optimal control, and shows that it is infeasible for our system. We consider optimal control of the flow control system described in §5.5. One optimal control technique that is popular in the control theoretic literature is Linear Quadratic Gaussian (LQG) control. This technique provides optimal control for a linear system where the goal is to optimize a quadratic metric in the presence of gaussian noise. We follow the description of LQG presented in [102] (pg. 835).

The quadratic performance index to minimize is given by

$$J = \mathbf{x}^T \mathbf{Q}\, \mathbf{x}$$

where the state vector **x** is modified so that it reflects a setpoint of n_b at $B/2$. Thus, minimizing the expected value of J will keep the state close to the setpoint, so that the system is close to optimal.

There is a problem with this formulation. Classical LQG demands that J include a term that minimizes u (the control effort). In our case, we are not interested in reducing u, since a) increasing u is not costly and b) in any case, the goal is to send at the maximum possible rate, and this corresponds to *maximizing u* rather than minimizing it! If we impose this restriction, then we can no longer do standard LQG.

We can get around this problem by modifying the criterion so that

$$J = \mathbf{x}^T\mathbf{Qx} + \mathbf{u}^T\varepsilon\mathbf{Iu}$$

and then considering the control as $\varepsilon \to 0$ (assume for the moment that the limit of the series converges to the value of the control at the limit). However, note that the Kalman criterion for stability of the optimal control is that:

$$rank\ [\mathbf{Q}^{\frac{1}{2}^*} \mid \mathbf{G}^*\mathbf{Q}^{\frac{1}{2}^*} \mid (\mathbf{G}^*)^2\mathbf{Q}^{\frac{1}{2}^*}] = 3 \qquad 5.B1$$

where $\mathbf{Q}^{\frac{1}{2}}$ is defined by $\mathbf{Q} = \mathbf{Q}^{\frac{1}{2}}\mathbf{Q}^{\frac{1}{2}}$, and the * denoted the conjugate transpose operator. If we want to minimize $(n_b - B/2)^2$, then

$$\mathbf{Q} = \begin{bmatrix} 1\,0\,0 \\ 0\,0\,0 \\ 00\,0 \end{bmatrix}$$

Thus, the rank of the matrix in (5.B1) is 1, which is less than 3. Hence, the Kalman stability criterion is not satisfied, and the LQG optimal control is *not stable*.

Note that the problem with the control is not due to our assumption about **R** being εI. Rather, this is because of the nature of the matrix **Q**. However, the nature of **Q** is determined completely by the form of **x** and the nature of the control problem itself. We conclude that for this system, stable LQG control is not feasible.

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 βrtt 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 δ insignificant, so the FQ algorithm was implemented with δ=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

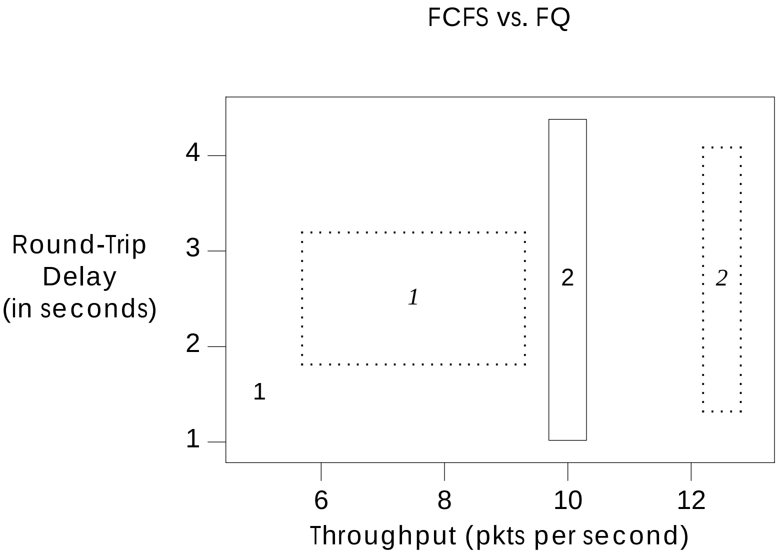


Figure 6.1: Example of results

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

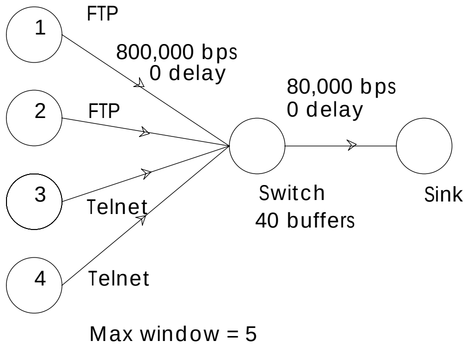


Figure 6.2: Scenario 1

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.

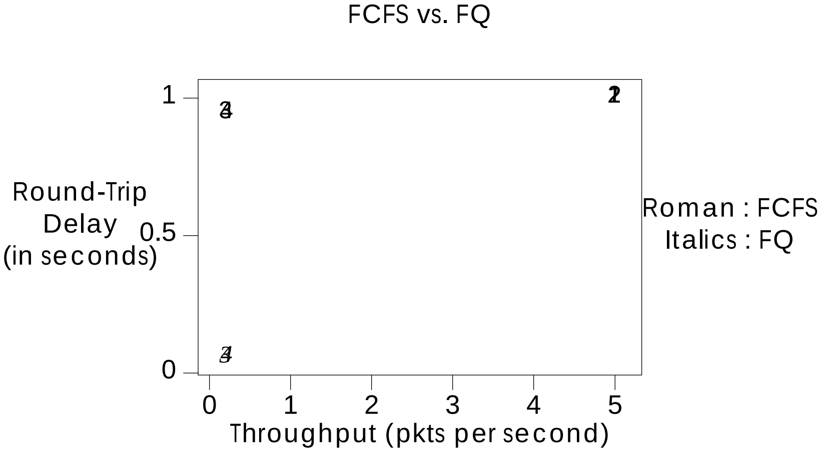


Figure 6.3: Scenario1: Utility diagram for generic flow control (G)

6.6.2. Scenario 2

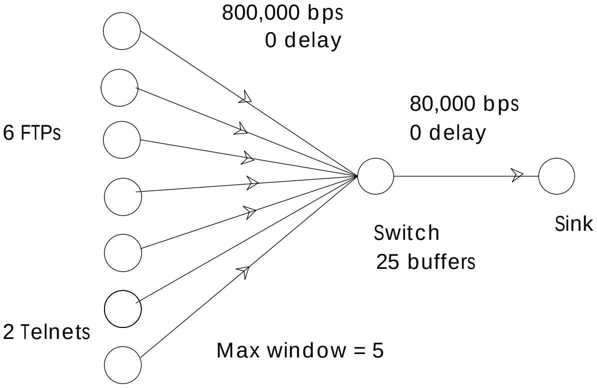


Figure 6.4: Scenario 2

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

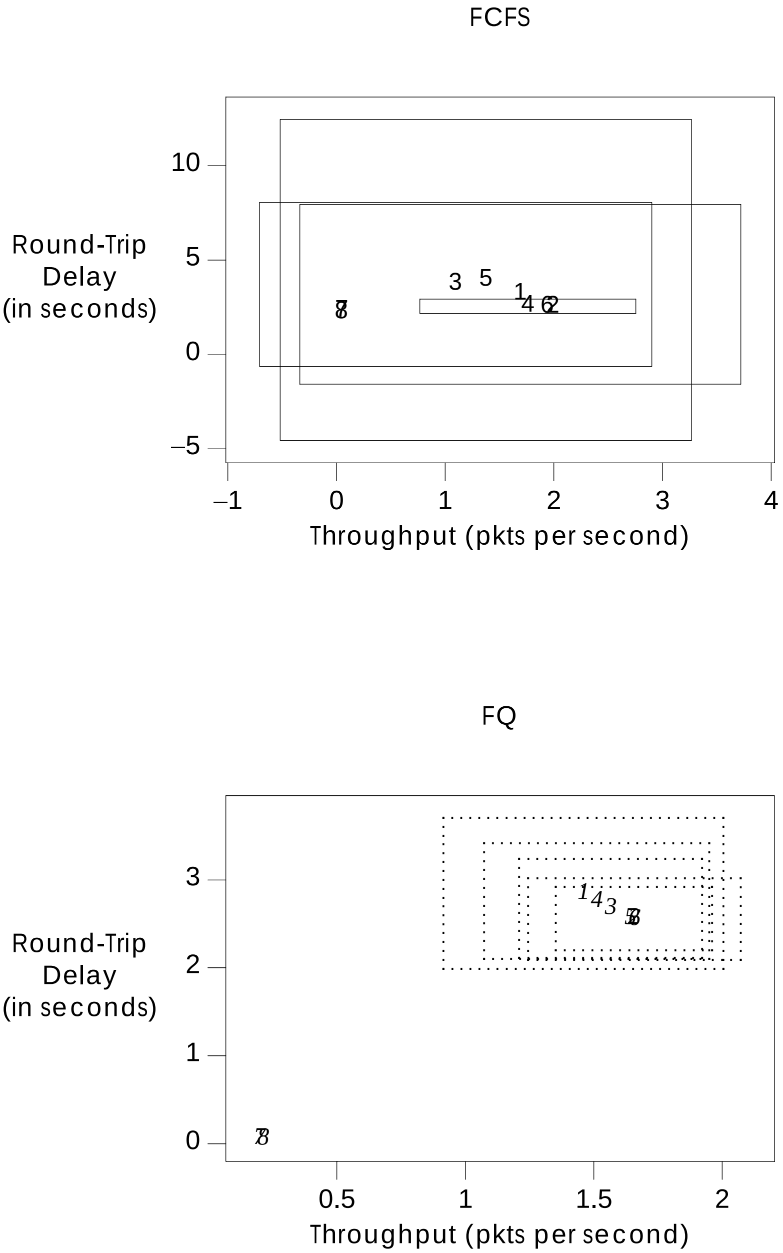


Figure 6.5 Scenario 2: Generic flow control: FCFS vs. FQ

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.

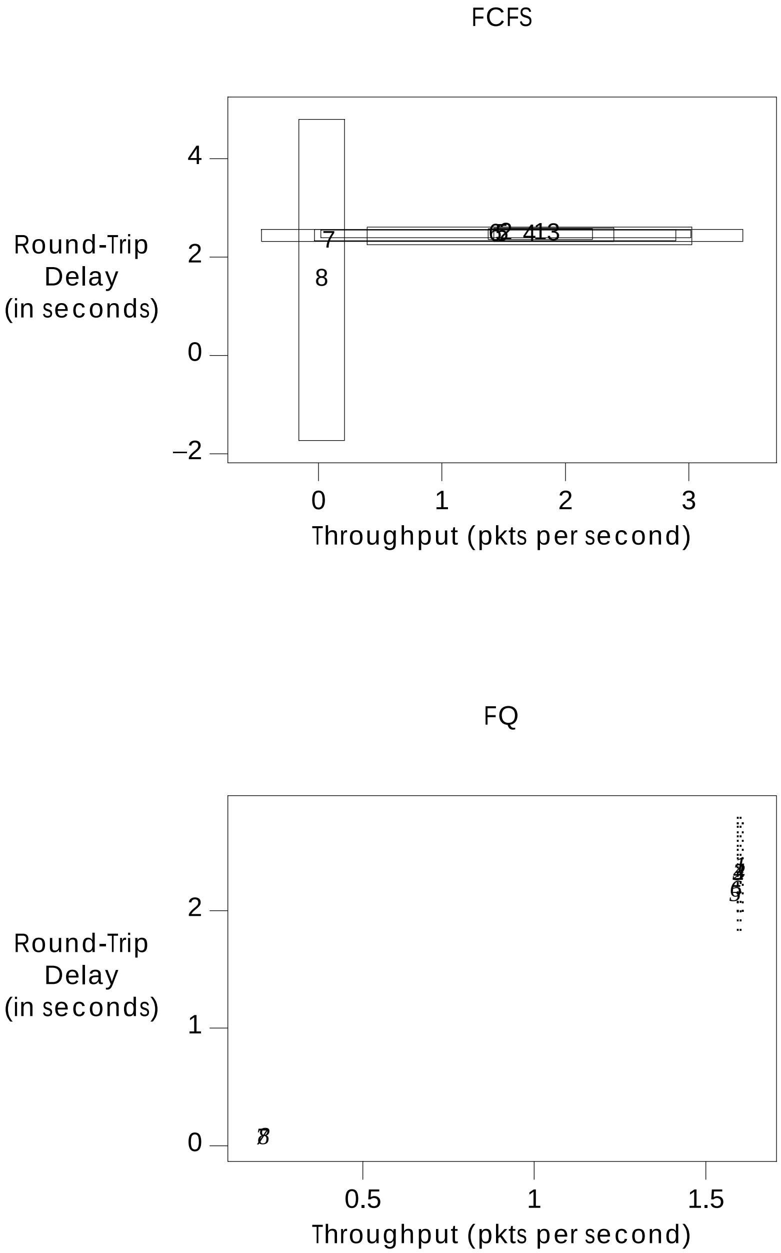


Figure 6.6: Scenario 2: JK flow control: FCFS vs. FQ

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

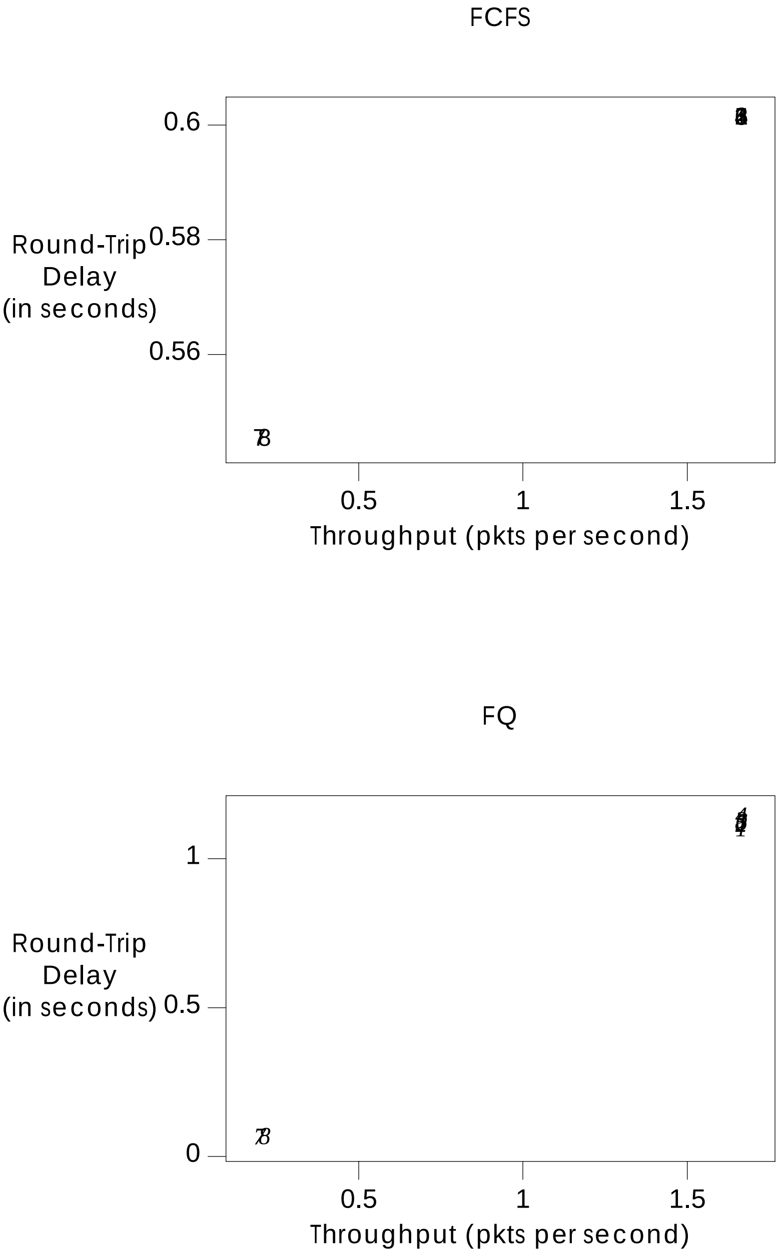


Figure 6.7: Scenario 2: DEC flow control: Selective DECbit vs. FQbit

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

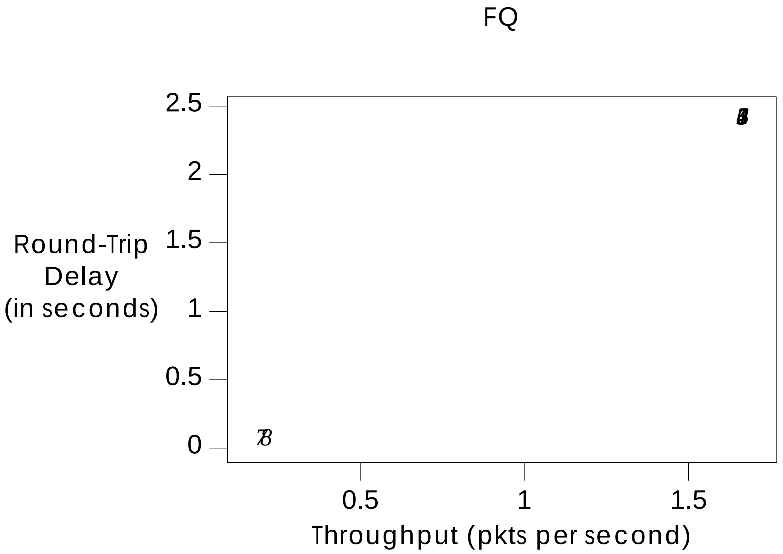
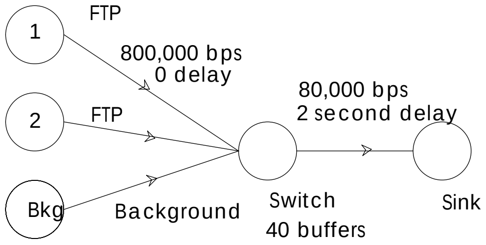


Figure 6.8: Scenario 2: PP_CTH flow control

overloaded networks.

6.6.3. Scenario 3



Max window size = 40

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.

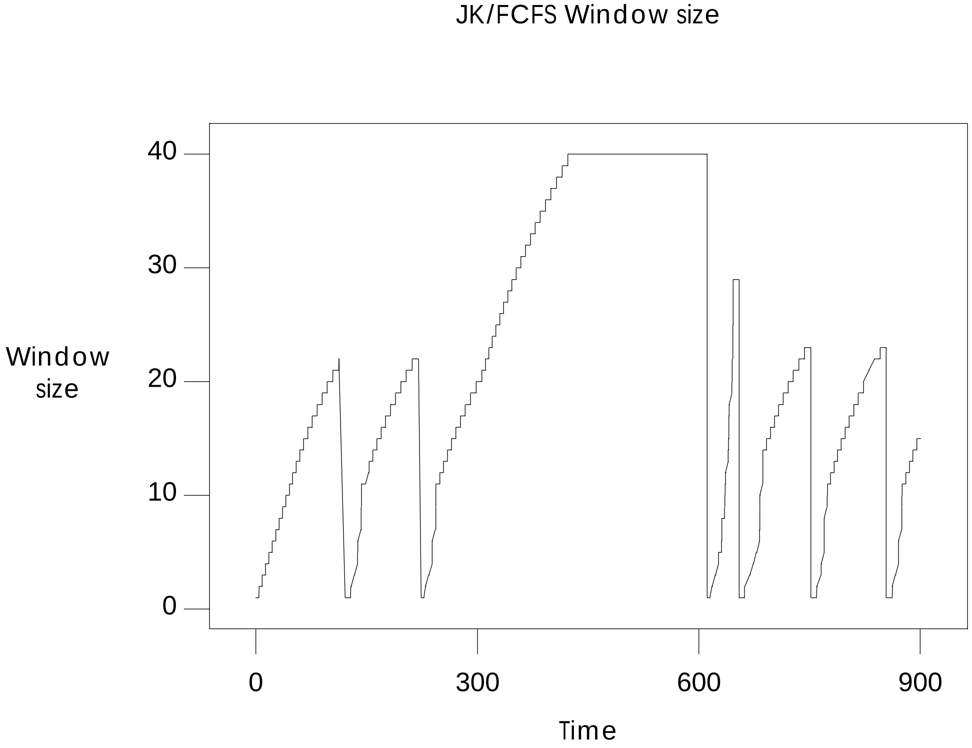


Figure 6.10: Scenario 3: JK/FCFS Window dynamics

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).

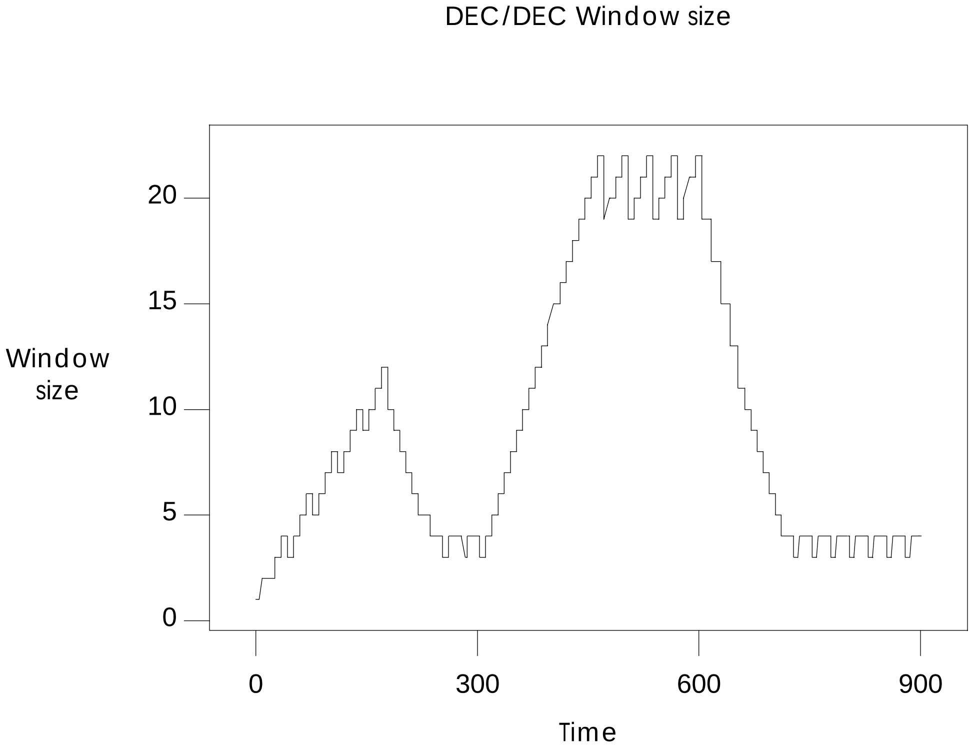


Figure 6.11: Scenario 3: DEC/DEC Window dynamics

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

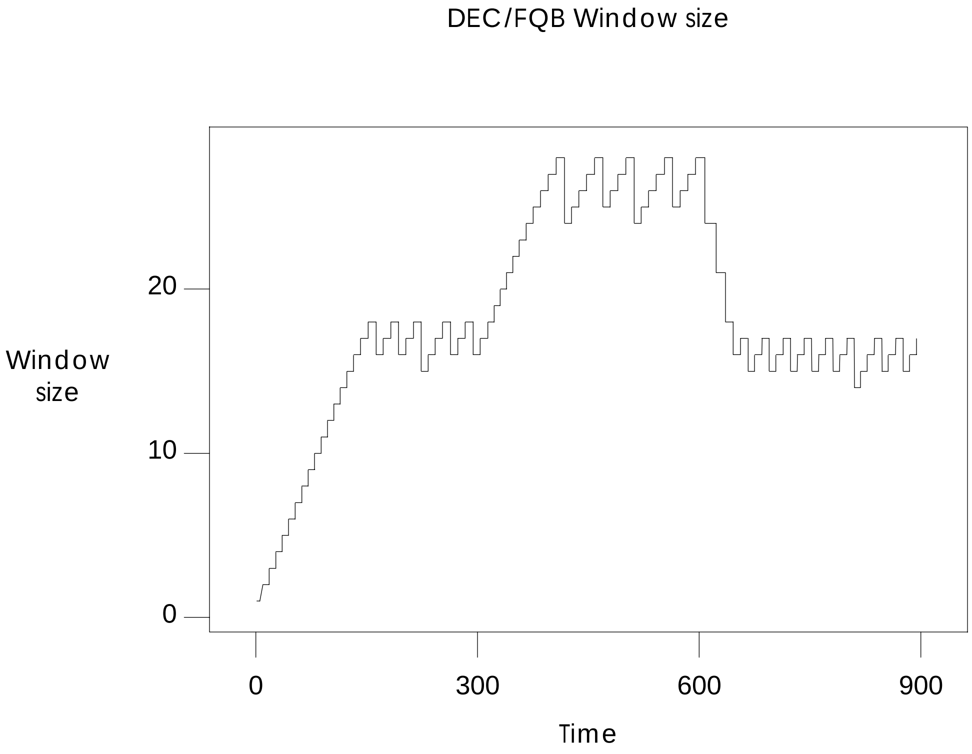


Figure 6.12: Scenario 3: DEC/FQB Window dynamics

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

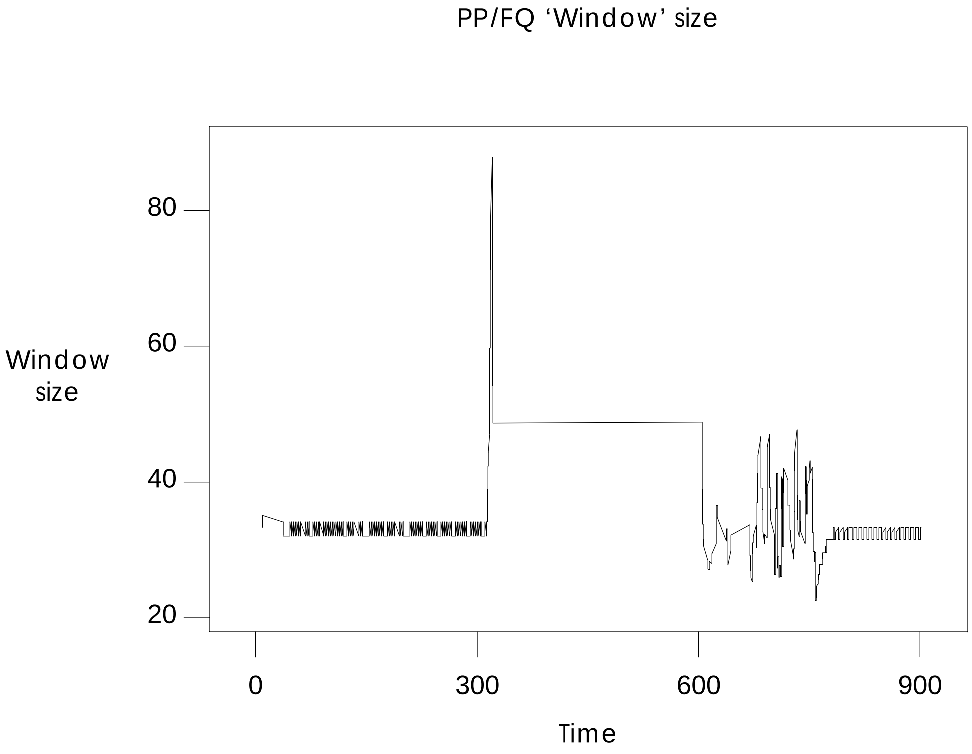


Figure 6.13: Scenario 3: PP/FQ 'Window' dynamics

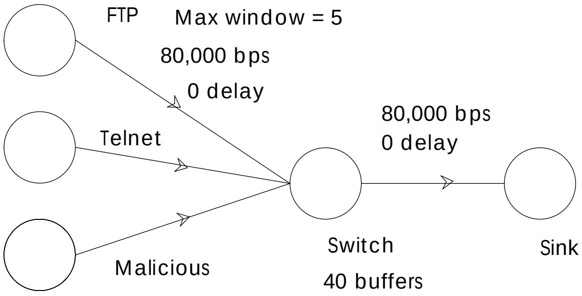


Figure 6.14: Scenario 4

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

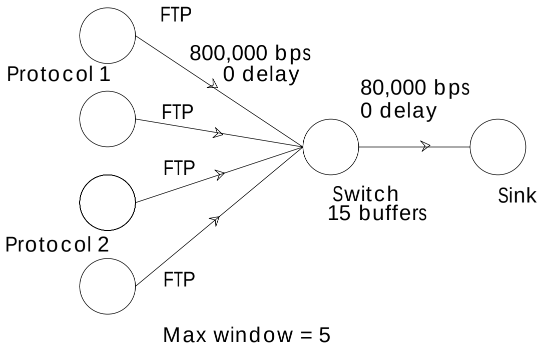


Figure 6.15: Scenario 5

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

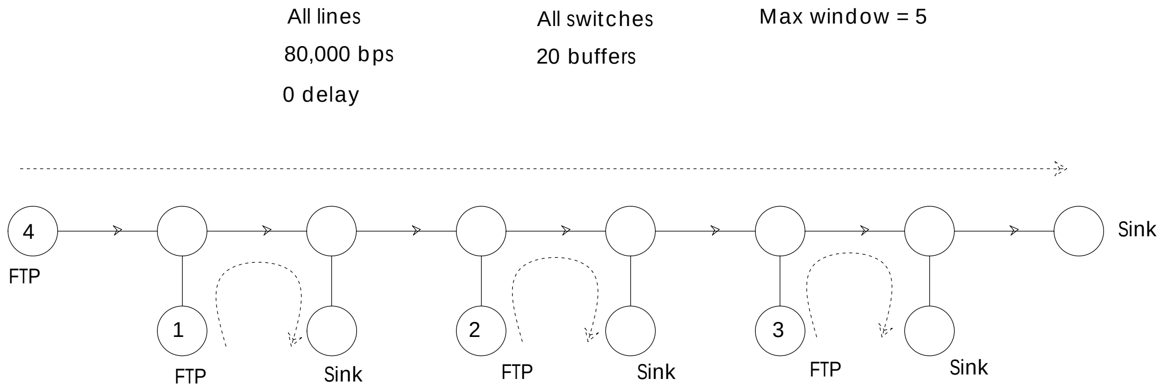


Figure 6.16: Scenario 6

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

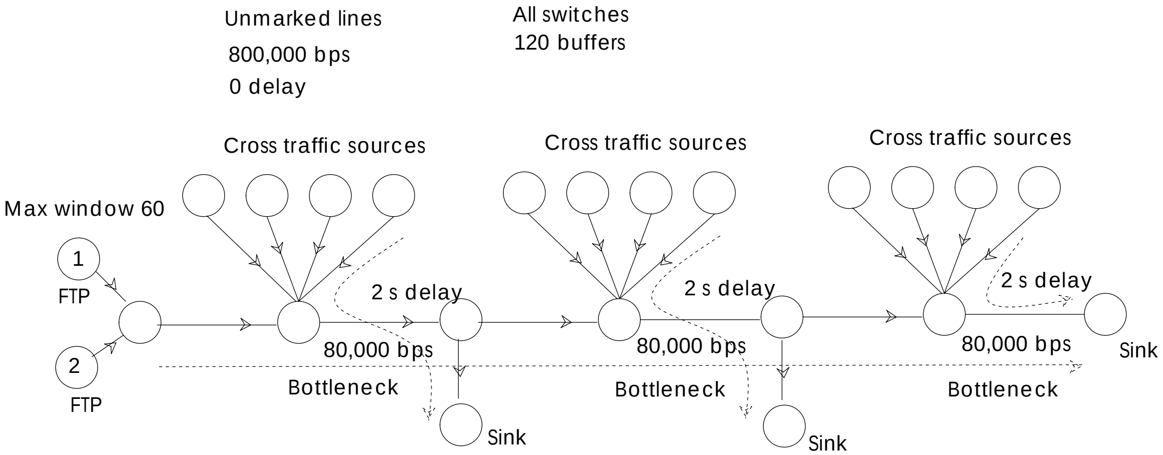


Figure 6.17: Scenario 7

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.

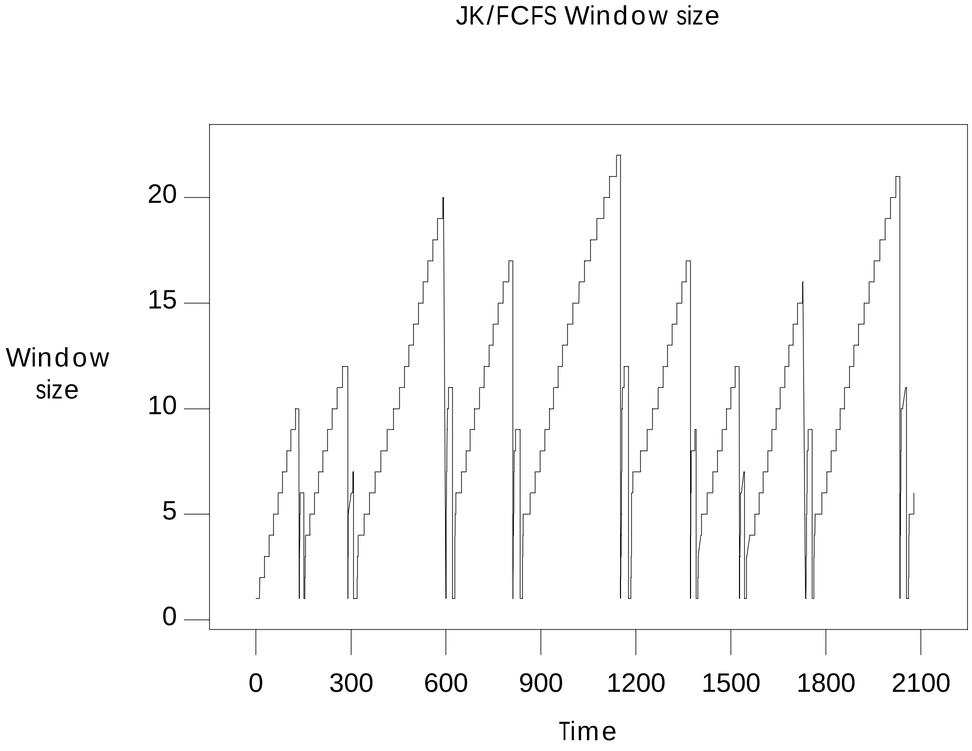


Figure 6.18: Scenario 7: JK/FCFS window vs. time

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).

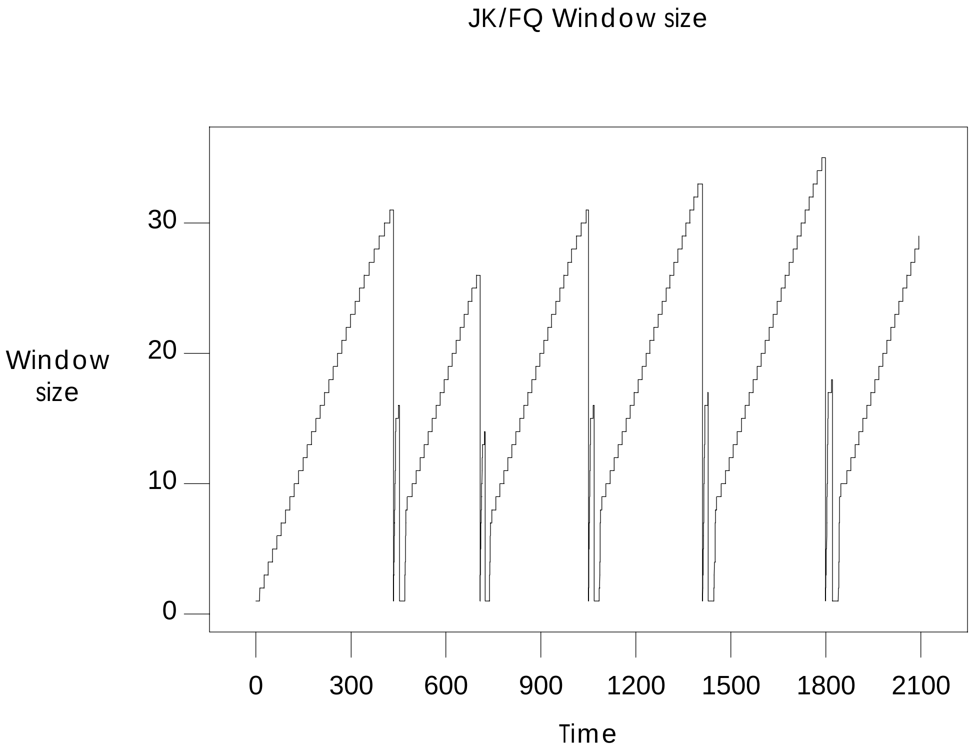


Figure 6.19: Scenario 7: JK/FQ window vs. time

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 α.

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 α 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

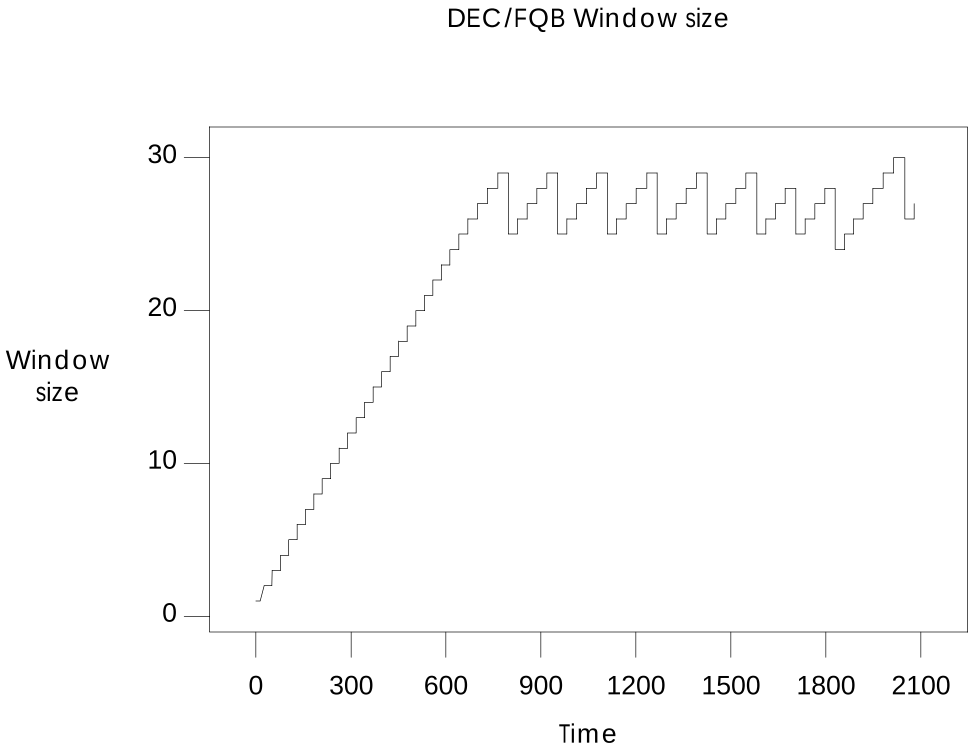


Figure 6.20: Scenario 7: DEC/FQB window vs. time

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.

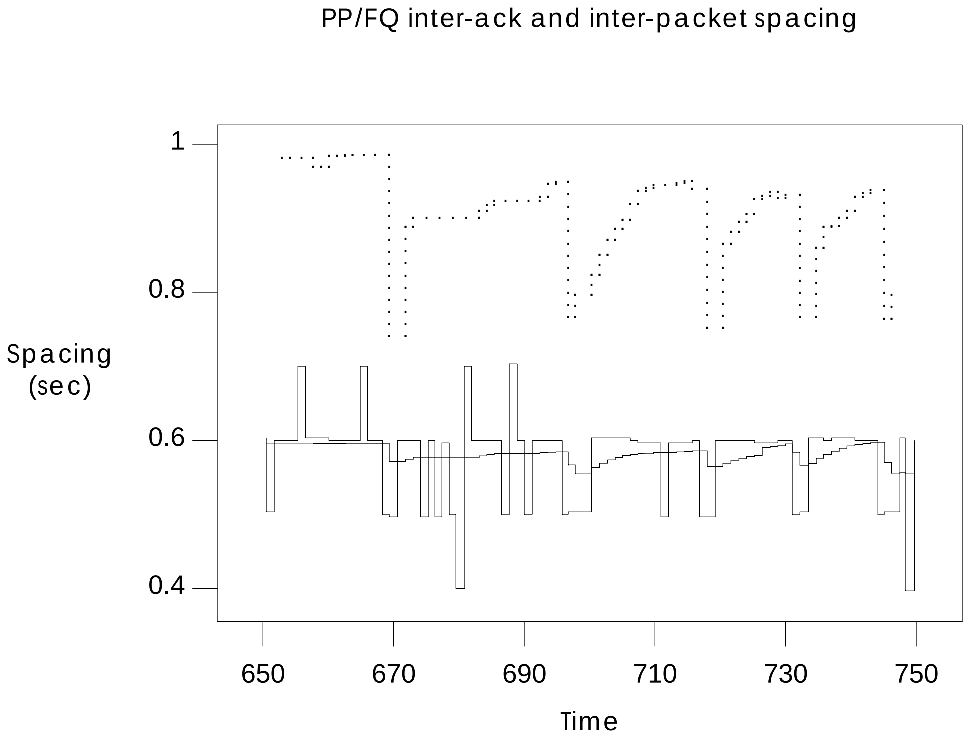


Figure 6.21: Scenario 7: PP/FQ inter-ack spacing, inter-packet spacing and α

6.6.8. Scenario 8

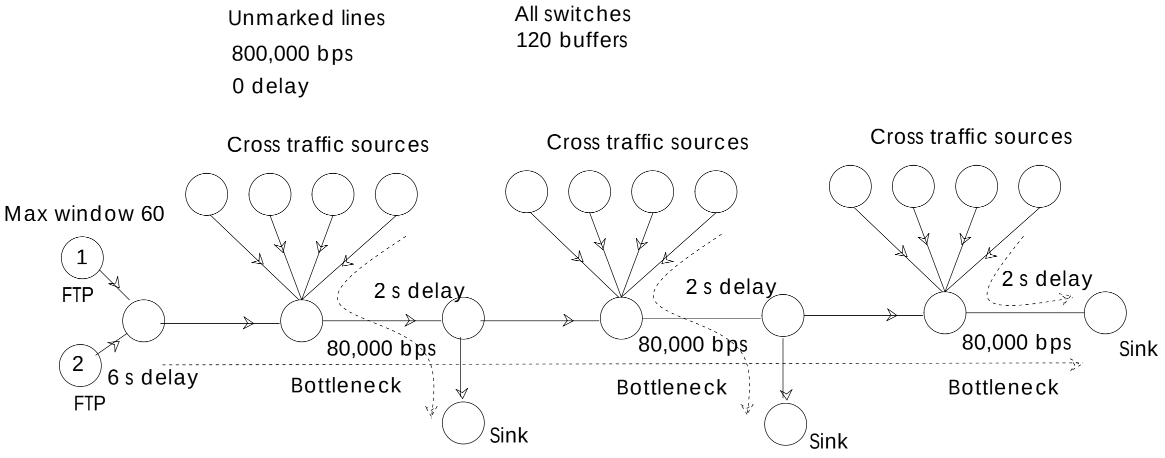


Figure 6.23: Scenario 8

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.

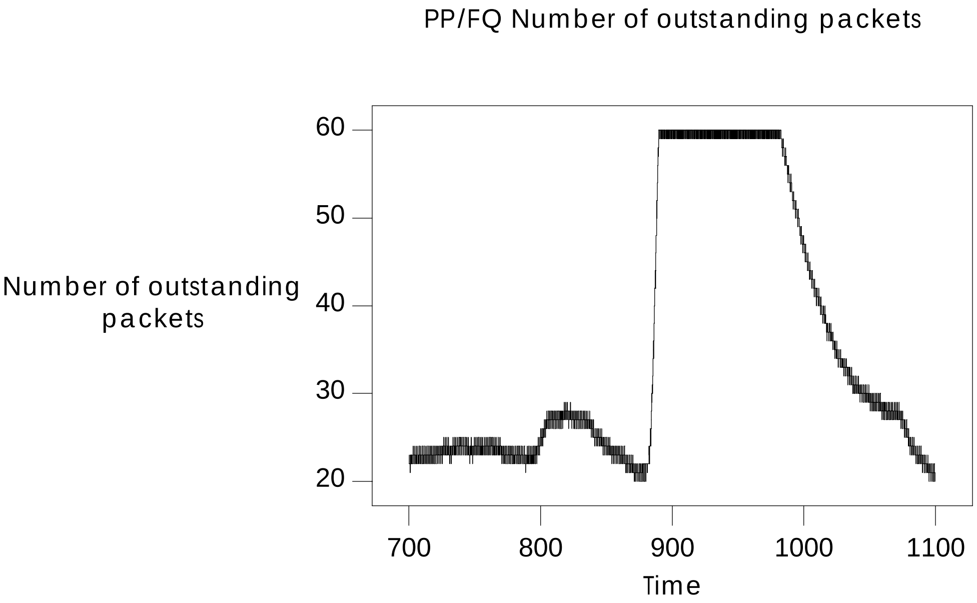


Figure 6.22: Scenario 7: Number of outstanding packets vs. time

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.

Chapter 7: Conclusions

7.1. Introduction

Chapter 1 presented a survey of congestion control techniques and a set of desirable characteristics of any congestion control scheme. In this chapter, we review the results from the preceding chapters, and examine the extent to which we have been successful in our design effort. We also examine the weaknesses in our work, and areas for future work.

7.2. Summary of the thesis

We define congestion as the loss of utility to a network client due to an overload in the network. This motivates the design of congestion control schemes, which allow users to gain utility from the network either by reserving resources to prevent overload, or by reacting to increased network loads. While congestion control can, and should, operate at a number of time scales concurrently, we restrict the scope of this thesis to reactive control schemes that operate at time scales ranging from less that one round trip time to a few round trip times. This corresponds to the design of a scheduling discipline that operates at all queueing points, and a transport level flow control protocol that is executed at all the hosts. In the course of the thesis, we describe and analyze the Fair Queueing scheduling discipline (FQ) and the Packet-Pair flow control protocol (PP), and claim that these mechanisms provide the required functionality and performance.

Chapter 2 introduced FQ as a way to provide a fair share of switch resources to competing conversations. We defined the notion of min-max fairness, and showed how this leads naturally to the FQ algorithm. A simple analysis considered the delay distribution of a Telnet conversation competing with FTP conversations at a FQ server. Chapter 3 studied data structures and algorithms for the efficient implementation of FQ, particularly the packet buffering scheme. We concluded that, if packet losses are few, then a simple ordered linked list is the best alternative. If there can be many losses, then a per-conversation linked list data structure is best. In Chapter 4, the traffic delinking property of FQ is used to build a deterministic model for a conversation in a network, and, from a series of lemmas, we derive the paired-packet probe. This is used to design the first version of the PP protocol, which constantly adjusts the data transmission rate to measured changes in the bandwidth-delay product. In Chapter 5 we recognize the inadequacies of the deterministic model, and propose a stochastic extension. This motivates a control theoretic approach to decide how best to use the series of packet pair probes to derive a stable flow control protocol. Practical issues, such as the unavailability of noise variances, motivated the design of a fuzzy prediction scheme. The resulting protocol allows users to obtain a desired delay-throughput tradeoff by choosing a setpoint, and dynamically adjusting the packet transmission rate so that the setpoint is maintained. As in all design problems, the proof of the pudding is in the eating. We showed in Chapter 6 that PP can match, or better, the performance of some widely known flow control schemes in a variety of scenarios, some of which are specifically designed to contradict our assumptions.

We now consider the role of FQ and PP as congestion control algorithms. The FQ scheduling discipline provides a number of advantages. First, it protects well-behaved users from ill-behaved ones, so that well-behaved users can get utility from the network even in the presence of ill-behaved or malicious users (this was the main purpose of the algorithm as proposed by Nagle [101]). Second, because it provides the equivalent of per-channel queueing, users can choose their own delay-throughput tradeoff. As described in §5.3.1, by choosing a setpoint for the queue size at the bottleneck, users can trade low delay for possible loss of throughput. Such tradeoffs are not possible with a FCFS discipline. (Even the selective DECbit protocol can only enforce a global tradeoff, because the FCFS discipline does not allow it to provide different users with different queueing delays.) Third, by partially delinking the traffic characteristics of the sources, FQ allows each user to probe the network state. This allows for sophisticated flow control that can use this information to choose an appropriate sending rate. Finally, FQ switches give incentives for sources to do more sophisticated flow control. As the simulation results for scenario 5 in Chapter 6 showed, sources that change to PP from JK or generic flow control get better performance.

The PP protocol leverages off FQ to do intelligent flow control. It too provides utility to users in several ways. First, it allows users to choose a bandwidth-delay tradeoff that corresponds to their utility function. A user's utility translates to a choice of setpoint, and PP ensures that the flow control tracks the setpoint to the extent allowed by control delays. Second, PP allows FTP sources to maintain their throughput even if the conversation has a large round trip propagation delay, or there is a lot of cross traffic. As we saw in scenario 8, the other popular flow control schemes we investigated do not perform too well under such circumstances. Finally, the protocol adapts quite rapidly to changes in the network state. Thus, even short periods where bandwidth is available at the bottleneck can be utilized.

Thus, both FQ and PP are successful congestion control schemes, in the sense that they allow a user to gain as much utility as possible from the network, even when it is overloaded.

7.3. Requirements re-examined

We now re-examine the seven requirements for a congestion control scheme as presented in Chapter 1, and see, using the analysis of Chapters 2-5 and the simulations of Chapter 6, to what extent these requirements have been met.

7.3.1. Efficiency

We desire a congestion control scheme to be efficient in two ways: not to consume excessive amounts of resources, such as network bandwidth and switch CPU time, and second, to allow the maximum possible utilization of network capacity.

PP does not place any overhead on the net amount of data transferred in the network, since additional probe or state-exchange packets are not used. Concerning the switch CPU time requirement, with PP, switches can be completely passive. Unlike the DECbit scheme, they need not set bits, nor compute averages. FQ implementation looks complicated at first glance, and it seems as if it may take a lot of switch CPU cycles. However, as we showed in Chapter 3, the additional overhead is not exceedingly large. With an efficient implementation scheme, we feel that the benefits of doing FQ outweigh its costs.

With the PP and FQ schemes, the network bandwidth is not underutilized (as it is, for example, with the DECbit schemes in Scenario 6). Since FQ is work conserving, it does not keep bandwidth idle if there are packets waiting to be served. Further, PP adjusts the packet transmission rate so that the bottleneck queue is never emptyl; hence a source obtains all the throughput allocated to it as its fair share. Indeed, we note that in Chapter 6, in all the scenarios, all the available bottleneck capacity is utilized.

Thus, we meet both the efficiency criteria mentioned in Chapter 1.

7.3.2. Heterogeneity

We require that the congestion control scheme accommodate heterogeneity in packet size, transport layer protocols and type of service requirements. FQ clearly allows for heterogeneity in packet sizes: indeed, that is one reason why we modified Round-Robin to obtain FQ. Second, by allowing users to choose their own delay-throughput tradeoffs, PP with FQ can accommodate a variety of service requirements. Finally, the results of scenario 5 indicate that PP can coexist with other protocols, and perform as well as, or better, than they can. Thus, we have satisfied our heterogeneity requirements.

The only caveat to the above is that, for PP to work correctly, every *potential* bottleneck must implement FQ or a similar round-robin-like service discipline. Even if a single bottleneck serves packets using FCFS, then the values reported by the probes will no longer be valid, and PP is no longer feasible. However, if the FCFS bottlenecks enforce some sort of rate control, and can stamp packets with the current service rate, then PP can be salvaged. This restriction is a major hurdle to the implementation of PP in current, FCFS networks.

7.3.3. Ability to deal with ill-behaved sources

Scenario 4 adequately answers this requirement.

7.3.4. Stability

In Chapter 5 we precisely defined the stability of a flow control protocol. Given the state equation that describes the dynamics of the length of the queue at the bottleneck, a flow control protocol is stable if the queue length remains bounded. This is true if the eigenvalues of the discrete time state equation lie in the unit circle, or the eigenvalues of the continuous time state equation lie in the left half plane. In the chapter, we formally proved the stability of PP. We are not aware of stability proofs for any other congestion control scheme in the literature.

7.3.5. Scalability

Scaling is required along two axes: bandwidth and network size. Higher bandwidths translate to larger bandwidth-delay products, so that sources can no longer ignore the propagation delay in doing flow control. Both our deterministic and our stochastic model explicitly model the control delay, and the PP scheme, which is based on these models, is designed to work in environments with large delays. The effectiveness of PP in networks with large propagation delays is seen in the results of scenarios 7 and 8, where, even with large control delays, PP sources behave well.

As the network increases in size, far more conversations are served at each switch. How well do our schemes cope with this? Chapter 3 showed how to implement efficient data structures to buffer data from many conversations. We believe that implementing these data structures in hardware is feasible. Special purpose hardware will allow FQ to serve a large number of conversations at high speeds.

A large number of conversations also means that the service rate fluctuations for any single conversation are smoothed out. PP is specifically designed for this environment, and, in fact, its performance will only improve as the number of users increases. Thus, we claim that our congestion control scheme scales well along both axes.

7.3.6. Simplicity

The simplicity requirement is that a congestion control scheme be easy to specify and implement. Both FQ and PP are conceptually simple, and can each be implemented in under two pages of C code. The control law of Chapter 5 translates to a single line of code, and the entire fuzzy controller takes about 20 lines of code.

One possible complication with PP as we presented it is that is requires one timer per packet. However, this can be changed to a single timer per conversation with little loss of performance. As a packet is transmitted, this timer is set to the packet's timeout value. When the timer expires, the last unacknowledged packet is retransmitted. With this scheme, the timeout interval is a little larger than the correct value, but there is a substantial savings in implementation cost.

FQ conceptually requires per-packet queueing, but, as we saw in Chapter 3, this can be implemented by a single ordered linked list. In any case, with hardware support, we expect that even per-conversation queueing can be implemented at high speeds (we are aware of one implementation that enqueues and dequeues ATM cells at 1.2 Gbps).

These figures, however, ignore the realities of protocol implementation, particularly in the Unix kernel. We are aware of the complexity of implementing a protocol at the kernel level, but there is nothing specific to PP or FQ that makes it any harder to implement in the kernel than any other equivalent protocol.

7.3.7. Fairness

FQ, as its name signifies, makes an effort to deliver min-max fairness to all the conversations that it serves. Fairness, however, requires that the flow control protocol be slightly sophisticated about managing its buffers. We saw that the generic flow control protocol does not allow for fair bandwidth allocation even with FQ. However, in all the simulated scenarios, the PP/FQ protocol pair provides fair (or nearly fair) bandwidth allocations to the sources. Thus, we claim that PP/FQ is a fair congestion control scheme.

We conclude that our schemes, with some minor caveats, satisfy the requirements raised in the first chapter. This is not true for a majority of the congestion control schemes in the literature.

7.4. Comments on design methodology

The success of our schemes owes mainly to our design methodology. We believe that, while one should not ignore practical problems, congestion control schemes must be based on a sound theoretical basis. This section presents the mapping from theory to practice that underlies our design effort.

Fair queueing is based on the principle of min-max fairness. The definition of min-max fairness immediately points to a bit-by-bit-round-robin (BR) scheme as a mechanism to obtain it, and FQ can be considered to be a practical implementation of this impractical ideal. This theoretical background makes FQ robust in the face of ill-behaved users.

The similarity of a BR scheme to time-division multiplexing motivates the deterministic model of Chapter 4. Once the model is stated, a little insight yields the packet-pair probe (this approach to the packet-pair scheme was first introduced by Prof. S. Singh and Prof. A. Agrawala [129]). This gives us a clean way to probe network state.

Having recognized that the network state may change, we see the need for a formal basis to express the dynamics of the network state. This basis is provided by control theory, and the control law used by PP is derived from a straightforward application of principles of predictive control.

The need for state estimation to implement control laws is a well known problem [49]. While Kalman estimators, and other related estimators, are theoretically adequate, it is increasingly recognized that, for a large class of practical applications, fuzzy logic has an important role to play. We use some fundamental principles of fuzzy control to build the fuzzy predictor described in Chapter 5. As the simulations in Chapter 6 show, this grounding in theory greatly helps the protocols in practice.

7.5. Contributions

This dissertation has made a number of contributions to the area of reactive congestion control. We review some of them in this section.

Our main contribution lies in the design of the Fair Queueing discipline and the Packet-Pair flow control protocol. (The design of FQ was joint work with S. Shenker and A. Demers.) We believe that both schemes, though conceptually simple, have several interesting features that will make them suitable for networks of the future. The two work together to provide better congestion control than some other schemes which are widely implemented in current networks.

Besides the schemes themselves, our other contributions are in developing deterministic and stochastic models for a conversation in a network, developing a control-theoretic approach to flow control in networks of FQ servers, and design of a fuzzy prediction algorithm.

Our use of deterministic modeling (which is joint work with Prof. S. Singh and Prof. A. Agrawala) makes the exact analysis of transient queueing phenomena possible. Though the model is naive, the stochastic version of the model allows for a formal control theoretic approach to flow control more easily than an equivalent stochastic queueing model.

Control theoretic approaches to flow control have been studied earlier for single M/M/1 servers and for Jacksonian networks. Our contribution lies in carrying out the analysis for a deterministic queueing model with propagation delays. The resulting protocol was shown to be stable. Further, we have implemented the resulting protocol in a realistic network simulator, and have done extensive simulations to study its behavior in a variety of benchmark scenarios.

We recognize the importance of state estimation in a distributed system with propagation delays. While we did derive the optimal Kalman estimator for the system state, we feel that such an approach is impractical for flow control protocols. Instead, our fuzzy prediction technique provides a practical alternative that performs well, and requires no additional information from the system.

7.6. Weaknesses and areas for future work

While we believe that our work has several claims to success, there are some weaknesses as well. Our theoretical models, though adequate for our purposes, are rather naive. We believe that even better results can be achieved by using more sophisticated models for the analysis. Thus, this thesis is only a small step in providing practical solutions to congestion control. What is heartening is that, even with these naive models, much can still be achieved.

Second, the simulations in Chapter 6, though extensive, are still far from a complete study of protocol behavior. While our choice of benchmark scenarios tries to test for several aspects of congestion control schemes, there are surely other aspects that we have overlooked. This is an area where much additional work is needed.

Finally, this thesis ignores two major dimensions of congestion control. To begin with, we do not consider predictive control. However, this is considered in detail in a contemporaneous thesis by D. Verma [141]. Also, we have ignored congestion control on time scales larger than multiple RTTs. Studying issues on larger time scales, and integrating the solutions into a single control scheme, is an avenue for future work. Other areas for future work are mentioned at the end of each chapter.

In conclusion, we feel that the area of congestion control is vast, and still in its infancy. We hope that this thesis makes a contribution to the field.

Bibliography

1. C. Agnew, Dynamic Modeling and Control of Congestion-prone Systems, *Operations Research 24*, 3 (1976), 400-419.
2. B. D. O. Anderson and J. B. Moore, *Optimal Filtering*, Prentice Hall, 1979.
3. B. D. O. Anderson and J. B. Moore, *Linear Quadratic Methods*, Prentice Hall, 1990.
4. D. Bacon, A. Dupuy, J. Schwartz and Y. Yemini, Nest : A Network Simulation and Prototyping Tool, *USENIX 88*, 1988.
5. J. J. Bae and T. Suda, Survey of Traffic Control Protocols in ATM Networks, *Proc. Globecom 1990*, December 1990, 300.1.1-300.1.6.
6. K. Bharath-Kumar and J. M. Jaffe, A New Approach to Performance-Oriented Flow Control, *IEEE Trans. on Communication COM-29*, 4 (April 1981), 427-435.
7. M. Boiteux, Peak Load Pricing, *Journal of Business 33* (April 1960), 157-179.
8. J. Bolot, Dynamical Behavior of Rate-Based Flow Control Mechanisms, Comp. Sci.-Tech. Rpt. 2279.1, University of Maryland, October 1989.
9. A. D. Bovopoulos and A. A. Lazar, Decentralized Algorithms for Optimal Flow Control, *Proc. 25th Allerton Conference on Communications Control and Computing*, October 1987. University of Illinois, Urbana-Champaign.
10. A. D. Bovopoulos and A. A. Lazar, Asynchronous Algorithms for Optimal Flow Control of BCMP Networks, Tech. Rpt. WUCS-89-10, Washington University, St. Louis, MO, February 1989.
11. R. Brown, Calendar Queues: A Fast O(1) Priority Queue Implementation for the Simulation Event Set Problem, *Communications of the ACM 31*, 10 (October 1988), 1220-1227.
12. 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.
13. R. Caceres,, Measurements of Wide Area Internet Traffic, Comp. Sci. Dept. Tech. Rpt. 89/550 , University of California, Berkeley, December 1989.
14. D. Cheriton, Sirpent: A High-Performance Internetworking Approach, *Proc. ACM SigComm 1989*, September 1989, 158-169.
15. D. Chiu and R. Jain, Analysis of Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks, *Computer Networks and ISDN Systems 17* (1989), 1-14.
16. I. Cidon and I. Gopal, Paris: An Approach to Integrated High-Speed Private Networks, *International Journal of Digital and Analog Cabled Systems 1* (1988), 77-85.
17. D. D. Clark, M. L. Lambert and L. Zhang, NETBLT: A Bulk Data Transfer Protocol, RFC-998, Network Working Group, March 1987.
18. D. D. Clark, V. Jacobson, J. Romkey and H. Salwen, An Analysis of TCP Processing, *IEEE Communications Magazine*, June 1989, 23-29.
19. D. D. Clark, Policy Routing in Internetworks, *Journal of Internetworking Research and Experience*, September 1990, 35-52.
20. R. Cocchi, D. Estrin, S. Shenker and L. Zhang, A Study of Priority Pricing in Multiple Service Class Networks, *Proc. ACM SigComm 1991*, September 1991.
21. D. Comer, in *Internetworking with TCP/IP Principles, Protocols and Architecture*, Prentice Hall, Englewood Cliffs, NJ, 1988.

22. D. W. Davies, The Control of Congestion in Packet Switching Networks, *IEEE Trans. Communications 20* (June 1972), 546-550.

23. 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..

24. B. T. Doshi and S. Dravida, Congestion Control for Bursty Data in High Speed Wide Area Packet Networks: In-Call Parameter Negotiations, Preprint, AT&T Bell Laboratories, Crawfords Corner Road, Holmdel NJ 07733, March 1991.

25. C. Douligeris and R. Mazumdar, An Approach to Flow Control in an Integrated Environment, CU-CTR-Tech. Rpt.-50, Columbia University, 1987.

26. C. Douligeris and R. Mazumdar, On Pareto Optimal Flow Control in a Multiclass Environment, *Proc. 25th Allerton Conference, University of Illinois*, October, 1987.

27. C. Douligeris and R. Majumdar, User Optimal Flow Control in an Integrated Environment, *Proc. of the Indo-US Workshop on Systems and Signals*, January 1988. Bangalore, India.

28. A. Dupuy, J. Schwartz, Y. Yemini and D. Bacon, NEST: A Network Simulation and Prototyping Testbed, *Communications of the ACM 33*, 10 (October 1990), 63-74.

29. A. A. Economides, P. A. Ioannou and J. A. Silvester, Adaptive Routing and Congestion Control for Window Flow Controlled Virtual Circuit Networks, *Proc. 27th Allerton Conference on Communications, Control and Computing University of Illinois* (1989).

30. A. E. Ekberg, D. T. Luan and D. M. Lucantoni, Bandwidth Management: A Congestion Control Strategy for Broadband Packet Networks: Characterizing the Throughput-Burstiness Filter, *Proc. ITC Specialist Seminar*, Adelaide, 1989, paper no. 4.4.

31. D. Estrin, Policy Requirements for Inter Administrative-Domain Routing, Request for Comments 1125, Network Working Group, November 1989 .

32. D. F. Ferguson, The Application of Microeconomics to the Design of Resource Allocation and Control Algorithms, *PhD thesis*, Columbia University, 1989.

33. D. Ferrari, in *Computer Systems Performance Evaluation*, Prentice Hall, Englewood Cliffs, NJ, 1978.

34. D. Ferrari and D. Verma, Buffer Space Allocation for Real-Time Channels in a Packet-Switching Network, International Comp. Sci. Institute Tech. Rpt. 90-022, Berkeley, June 1990.

35. D. Ferrari, Client Requirements for Real-Time Communications Services, *IEEE Communications Magazine 28*, 11 (November 1990).

36. D. Ferrari and D. Verma, Quality of Service in ATM Networks, International Comp. Sci. Institute Tech. Rpt. 90-064, Berkeley, November 1990.

37. D. Ferrari and D. Verma, A Scheme for Real-Time Channel Establishment in Wide-Area Networks, *IEEE J. on Selected Areas in Communications*, April 1990.

38. J. Filipiak, *Modelling and Control of Dynamic Flows in Communication Networks*, Springer-Verlag, 1988.

39. S. Floyd and V. Jacobson, Traffic Phase Effects in Packet-Switched Gateways, *Computer Communications Review 21*, 2 (April 1991).

40. S. Floyd and V. Jacobson, On Traffic Phase Effects in Packet-Switched Gateways, Preprint, April 1991.

41. A. G. Fraser, Towards a Universal Data Transport System, *IEEE Journal on Selected Areas in Communication SAC-1*, 5 (Nov. 1983), 803-816.

42. A. Fraser and S. Morgan, Queueing and Framing Disciplines for a Mixture of Data Traffic Types, *AT&T Bell Laboratories Technical Journal 63*, 6 (1984), 1061-1087.

43. E. Gafni and D. Bertsekas, Dynamic Control of Session Input Rates in Communication Networks, *IEEE Trans. on Automatic Control 29*, 11 (1984), 1009-1016.

44. M. Gerla and L. Kleinrock, Flow Control : A Comparative Survey, *IEEE Trans. on Communication COM-28*, 4 (April 1980), 553-574.

45. A. Giessler, A. Jagemann, E. Maser and J. D. Hanle, Flow Control Based on Buffer Classes, *IEEE Trans. on Communication COM-29*, 4 (April 1981), 436-443.

46. S. J. Golestani, Congestion-Free Transmission of Real-Time Traffic in Packet Networks, *Proc. Infocom 1990*, June 1990, 527-536.

47. S. J. Golestani, A Stop-and-Go Queueing Framework for Congestion Management, *Proc. ACM SigComm 1990*, September 1990, 8-18.

48. S. J. Golestani, Duration-limited Statistical Multiplexing of Delay-Sensitive Traffic in Packet Networks, *Proc. Infocom 1991*, April 1990.

49. G. C. Goodwin and K. S. Sin, *Adaptive Filtering Prediction and Control*, Prentice Hall, 1984.

50. A. Greenberg and N. Madras, How Fair is Fair Queueing?, *Proc. Performance 90*, 1990.

51. R. Gusella, A Measurement Study of Diskless Workstation Traffic on an Ethernet, *IEEE Trans. on Communications*, September 1990.

52. R. Gusella, Characterizing the Variability of Arrival Processes with Indices of Dispersion, *IEEE J-SAC 9*, 2 (February 1991), 203-211.

53. Z. Haas, Performance of the Adaptive Admission Congestion Control Scheme, Preprint, AT&T Bell Laboratories, Crawfords Corner Road, Holmdel NJ 07733, March 1991.

54. 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.

55. E. L. Hahne, Round Robin Scheduling for Fair Flow Control in Data Communication Networks, LIDS-TH-1631, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139.

56. E. Hashem, Analysis of Random Drop for Gateway Congestion Control, LCS-Tech. Rpt. 465, Laboratory for Computer Science, Massachussetts Institute of Technology, Cambridge, MA, 1989.

57. A. T. Heybey and J. R. Davin, A Simulation Study of Fair Queueing, *Computer Communications Review 20*, 5 (October 1990), 23-29.

58. E. Horowitz and S. Sahni, *Fundamentals of Data Structures* , Prentice Hall, 1981.

59. M. Hsiao and A. A. Lazar, A Game Theoretic Approach to Decentralized Flow Control of Markovian Queueing Networks, *Proc. Performance '87*, Brussels, Belgium, December 1987, 55-73.

60. M. Hsiao and A. A. Lazar, Optimal Flow Control of Multi-Class Queueing Networks with Partial Information, *IEEE Transactions on Automatic Control 35*, 7 (July 1990), 855-860.

61. M. Irland, Simulation of CIGALE 1974, *Proc. ACM-IEEE 4th Data Commcn. Symp.*, P.Q., Canada, Oct. 1975.

62. M. I. Irland, Buffer Management in a Packet Switch, *IEEE Trans. on Communication COM- 26* (March 1978), 328-337.

63. V. Jacobson, Congestion Avoidance and Control, *Proc. ACM SigComm* , August 1988, 314-329.

64. R. Jain and K. K. Ramakrishnan, Congestion Avoidance in Computer Networks with a Connectionless Network Layer : Concepts, Goals and Methodology, *Proc. IEEE 1988 Computer Communication Conference*, August, 1988.

65. R. Jain, A Comparison of Hashing Schemes for Address Lookup in Computer Networks , Tech. Rpt.-593, Digital Equipment Corporation , February 1989.

66. R. Jain, A Delay-based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks, *Computer Communications Review*, October 1989, 56-71.

67. R. Jain, Myths About Congestion Management in High-Speed Networks, Technical Report-726, Digital Equipment Corporation, October 1990.

68. C. R. Kalmanek, H. Kanakia and S. Keshav, Rate Controlled Servers for Very High Speed Networks, *Proc. Globecom 1990*, December 1990, 300.3.1-300.3.9.

69. C. R. Kalmanek, Xunet 2: A Nationwide Testbed in High-Speed Networking, *Comp. Sci. Tech. Rpt.*, March 1991, AT&T Bell Labs, 600 Mountain Ave. Murray Hill, NJ 07974.

70. F. Kamoun, A Drop and Throttle Flow Control Policy for Computer Networks, *IEEE Trans. on Communication COM-29*, 4 (April 1981), 444-452.

71. P. Karn and C. Partridge, Improving Round-Trip Time Estimates in Reliable Transport Protocols, *Proc. ACM SigComm* , 1987, 2-7.

72. M. G. H. Katavenis, Fast Switching and Fair Control of Congested Flow in Broadband Networks, *IEEE JSAC SAC-5*, 8 (October 1987).

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

74. S. Keshav, REAL Manuals, Comp. Sci. Dept. Tech. Rpt. 89/530 , University of California, Berkeley, September 1989.

75. S. Keshav, The Packet Pair Flow Control Protocol, Tech. Rpt. 91-028, International Comp. Sci. Institute , Berkeley, CA 94704, May 1991.

76. S. Keshav and P. S. Khedkar, Fuzzy Prediction, Preprint, Comp. Sci. Division, Dept. of EECS, Univ. California, Berkeley, CA 94720., April 1991.

77. S. Kheradpir, PARS: A Predictive Access-Control and Routing Strategy for Real-Time Control of Telecommunication Networks, Unpublished GTE Report, GTE Laboratories Inc., 40 Sylvan Road, Waltham, MA 02254, 1988.

78. D. Knuth, Fundamental Algorithms, *Addison-Wesley*, 1973.

79. K. Ko, P. P. Mishra and S. K. Tripathi, Predictive Congestion Control in High-Speed Wide-Area Networks, in *Protocols for High Speed Networks II*, Elsevier Science Publishers/North-Holland, April 1991.

80. J. Laffont, Massachussetts Institute of Technology Press, Cambridge, 1988.

81. S. S. Lam and M. Reiser, Congestion Control of Store and Forward Networks by Buffer Input Limits, *Proc. Nat. Telecommncn. Conf.*, Los Angfeles, CA, Dec 1977.

82. G. Langari, Analysis and Design of Fuzzy Control Systems, *PhD thesis (in preparation)*, University of California, Berkeley, 1991.

83. A. A. Lazar, A. T. Temple and R. Gidron, MAGNET II: A Metropolitan Area Network Based on Asynchronous Time Sharing, *IEEE Journal on Selected Areas in Communications 8*, 8 (October 1990).

84. B. Leiner, Critical Issues in High Bandwidth Networking, Request for Comments 1077, Network Working Group, November 1988.

85. C. Lemieux, Theory of Flow Control in Shared Networks and Its Application in the Canadian Telephone Network, *IEEE Trans. on Communication COM-29*, 4 (April 1981), 399-413.

86. C. Lo, Performance Analysis and Application of a Two-Priority Packet Queue, *AT&T Technical Journal 66*, 3 (1987), 83-99.

87. D. Luan and D. Lucantoni, Throughput Analysis of an Adaptive Window-based Flow Control Subject to Bandwidth Management, *Proc. 12th International Teletraffic Conference*, 1988, 1062-1068.

88. J. C. Majithia, M. Irland, J. L. Grange, N. Cohen and C. O'Donnell, Experiments in Congestion Control Techniques , *Proc. Int. Symp. Flow Control Computer Networks*, Versailles, France, Feb. 1979, 211-234.

89. A. Mankin and K. K. Ramakrishnan, Performance and Congestion Control Working Group Report, *Internet Engineering Task Force Meeting*, July 1989.

90. A. Mankin, Random Drop Congestion Control, *Proc. ACM SigComm 1990*, September 1990.

91. J. Matsumoto and H. Mori, Flow Control in Packet-Switched Networks by Gradual Restrictions of Virtual Calls, *IEEE Trans. on Communication COM-29*, 4 (April 1981), 466-473.

92. N. F. Maxemchuck, Dispersity Routing in Store and Forward Networks, *PhD thesis*, University of Pennsylvania, May 1975.

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

94. D. Mills and W. Braun, The NSFNET backbone Network, *Proc. ACM SigComm 1987*, 1987, 191-196.

95. D. Mills, The Fuzzball, *Proc. ACM SigComm 1988*, 1988, 115-122.

96. 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.

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

98. S. P. Morgan, Queueing Disciplines and Passive Congestion Control in Byte-Stream Networks, *Proc. IEEE INFOCOM '89*, 1989, 711-729.

99. A. Mukherjee and J. C. Strikwerda, Analysis of Dynamic Congestion Control Protocols - A Fokker-Planck Approximation, *Proc. ACM SigComm '91*, September 1991.

100. G. J. Murakami, R. H. Campbell and M. Faiman, Pulsar: Non-Blocking Packet Switching with Shift-register Rings, *Proc. ACM SigComm 1990*, September 1990, 145-155.

101. J. Nagle, On Packet Switches with Infinite Storage, *IEEE Trans. on Communications COM-35* (1987), 435-438.

102. K. Ogata, Discrete Time Control Systems, *Prentice Hall*, 1987.

103. A. K. Pawlikowski, Steady State Simulation of Queueing Processes: A Survey of Problems and Solutions, *ACM Computing Surveys 22*, 2 (June 1990), 123-171.

104. P. F. Pawlita, Traffic Measurements in Data Networks, Recent Measurement Results, and Some Implications, *IEEE Trans. on Communications 29*, 4 (April 1981).

105. V.,Paxson, Measurements and Models of Wide Area TCP Conversations, LBL-30840, Lawrence Berkeley Laboratory, Berkeley, CA, June 1991.

106. E. Pazner, Pitfalls in the Theory of Fairness, in *Social Goals and Social Organization : Essays in the Memory of Elisha Pazner*, Cambridge University Press, New York, 1985.

107. E. Pazner, Recent Thinking on Economic Justice, in *Social Goals and Social Organization : Essays in the Memory of Elisha Pazner*, Cambridge University Press, New York, 1985.

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

109. J. Postel, Internet Protocol, Request for Comments 791, Network Working Group, 1981.

110. L. Pouzin, Methods, Tools and Observations on Flow Control in Packet-Switched Data Networks, *IEEE Trans. on Communication COM-29*, 4 (April 1981), 413-426.

111. I. Pressman, A Mathematical Formulation of the Peak Load Pricing Problem, *Bell Journal of Economics and Management Science 1* (Autumn 1970), 304-326.

112. W. Prue and J. Postel, Something a Host Could Do with Source Quench : The Source QUench Introduced Delay (SQUID), Request for Comments 1016, Network Working Group, July 1987.

113. W. Prue and J. Postel, A Queueing Algorithm to Provide Type-of-Service for IP Links, Request for Comments 1046, Network Working Group, 1988.

114. K. K. Ramakrishnan, Analysis of a Dynamic Window Congestion Control Protocol in Heterogenous Environments Including Satellite Links, *Proc. 1986 IEEE Symp. on Computer Networks*, 1986.

115. K. K. Ramakrishnan, D. Chiu and R. Jain, Congestion avoidance in Computer Networks with a Connectionless Network Layer - Part IV - A Selective Binary Feedback Scheme for General Topologies, Technical Report-510, Digital Equipment Corporation, November 1987.

116. K. K. Ramakrishnan and R. Jain, Congestion avoidance in Computer Networks with a Connectionless Network Layer - Part II - An Explicit Binary Feedback Scheme, Technical Report-508, Digital Equipment Corporation, April 1987.

117. 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.

118. P. V. Rangan, Trust Relationships, Naming, and Secure Communication in Large Distributed Computer Systems, Comp. Sci. Dept. Tech. Rpt. 88/456 , University of California, Berkeley, October 1988.

119. T. G. Robertazzi and A. A. Lazar, On the Modeling and Optimal Flow Control of the Jacksonian Network, *Performance Evaluation 5* (1985), 29-43.

120. K. K. Sabnani and A. N. Netravali, A High Speed Transport Protocol for Datagram/Virtual Circuit Networks, *Proc. ACM SigComm 1989*, September 1989, 146-157.

121. B. A. Sanders, A Public Good/Private Good Decomposition for Optimal Flow Control of an M/M/1 Queue, *IEEE Trans. on Automatic Control 30*, 11 (November 1985), 1143-1145.

122. B. A. Sanders, An Asynchronous, Distributed Flow Control Algorithm for Rate Allocation in Computer Networks, *IEEE Trans. on Computers 37*, 7 (July 1988).

123. B. A. Sanders, An Incentive Compatible Flow Control Algorithm for Rate Allocation in Computer Networks , *IEEE Trans. on Computers 37*, 9 (September 1988).

124. S. Shenker, Comments on the IETF Performance and Congestion Control Working Group Draft on Gateway Congestion Control Policies, Unpublished, 1989.

125. S. Shenker, Making Greed Work in Networks: A Game-Theoretic Analysis of Gateway Service Disciplines, Preprint, Xerox PARC, 3333 Coyote Hill Road, Palo Alto, CA 94304., September 1989.

126. S. Shenker, A Theoretical Analysis of Feedback Flow Control, *Proc. ACM SigComm 1990,* September 1990, 156-165.

127. 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.

128. M. Sidi, W. Liu, I. Cidon and I. Gopal, Congestion Control Through Input Rate Regulation, *Proc. Globecom 89,* December 1989, 1764-1768.

129. 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.

130. P. O. Steiner, Peak Loads and Efficient Pricing, *Quarterly Journal of Economics 71* (November 1957), 585-610.

131. S. Tan, An Architecture for Call Processing, Internal Choices Report, Dept. of Comp. Sci., University of Illinois at Urbana-Champaign, November 1990.

132. A. S. Tanenbaum, in *Computer Networks,* Prentice Hall, Englewood Cliffs, NJ, 1981.

133. D. Tipper and M. K. Sundareshan, Numerical Methods for Modeling Computer Networks under Nonstationary Conditions, *JSAC 8,* 9 (December 1990).

134. C. Topolcic, editor, Experimental Internet Stream Protocol, Version 2 (ST-II), RFC-1190, Network Working Group, October 1990.

135. S. Tripathi and A. Duda, Time-dependent Analysis of Queueing Systems, *INFOR 24,* 3 (1978), 334-346.

136. J. Turner, New Directions in Communications (or Which Way to the Information Age?), *IEEE Communication Magazine 24,* 10 (October 1986).

137. F. Vakil and A. A. Lazar, Flow Control Protocols for Integrated Networks with Partially Observed Traffic, *IEEE Transactions on Automatic Control 32,* 1 (1987), 2-14.

138. F. Vakil, M. Hsiao and A. A. Lazar, Flow Control in Integrated Local Area Networks, *Performance Evaluation 7,* 1 (1987), 43-57.

139. H. R. Varian, Equity, Envy, and Efficiency, *J. Econ. Theory 9* (1974), 63-91.

140. H. Varian, in *Microeconomic Analysis,* W.W. Norton and Company, 1978.

141. D. Verma, Guaranteed Performance Communication in High-Speed Networks, *PhD thesis,* University of California, December 1991.

142. D. Verma, H. Zhang and D. Ferrari, Delay Jitter Control for Real-Time Communication in a Packet Switching Network, *Proc. TriComm '91,* April 1991.

143. J. G. Waclawsky and A. K. Agrawala, Transfer Time and Queue Dynamics of Window Protocols, Comp. Sci.-Tech. Rpt.-2321, University of Maryland, September 1989.

144. J. G. Waclawsky and A. K. Agrawala, Dynamic Behavior of Data Flow within Virtual Circuits, Comp. Sci.-Tech. Rpt.-2250, University of Maryland , May 1989.

145. J. G. Waclawsky, Window Dynamics, *PhD Thesis,* University of Maryland, College Park, May 1990.

146. C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart and S. Stornetta, SPAWN: A Distributed Computational Economy, SSL-89-18, Xerox PARC, Palo Alto, CA, November 1990.

147. Z. Wang and J. Crowcroft, A New Congestion Control Scheme: Slow Start and Search, Preprint, University College, London, UK, October 1990.

148. C. L. Williamson and D. R. Cheriton, Loss-Load Curves: Support for Rate-Based Congestion Control in High-Speed Datagram Networks, *Proc. ACM SigComm 1991,* September 1991.

149. J. W. Wong, J. P. Sauve and J. A. Field, A Study of Fairness in Packet-Switching Networks, *IEEE Trans. on Communication COM-30*, 2 (Feb., 1982), 346-353.

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

151. L. A. Zadeh, Fuzzy Sets, *Journal of Information and Control 8* (1965), 338-353.

152. L. A. Zadeh, Outline of a New Approach to the Analysis of Complex Systems and Decision Processes, *IEEE Trans. on Systems, Man and Cybernetics*, 1973, 28-44.

153. L. Zhang, Why TCP Timers Don't Work Well, *Proc. Sigcomm 1986*, 1986, 397-405.

154. L. Zhang, A New Architecture for Packet Switching Network Protocols, *PhD thesis*, Massachusetts Institute of Technology, July 1989.

155. L. Zhang and D. Estrin, Design Considerations for Usage Accounting, *Computer Communications Review 20*, 5 (October 1990), 56-67.

156. H. Zhang and S. Keshav, Comparison of Rate-Based Service Disciplines, *Proc. ACM SigComm 1991*, September 1991. also International Comp. Sci. Institute Tech. Rpt. 91-024, Berkeley, CA..

157. 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.

158. H. J. Zimmerman, in *Fuzzy Set Theory and its Applications*, Kluwer Academic Publishers, 1985.