4

SECURITY CLASSIFICATION OF THIS PAGE

DTIC ELECTE JAN 02 1990 S B D

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) MIT/LCS/TR-455	5 MONITORING ORGANIZATION REPORT NUMBER(S) N00014-83-K-0125
6a NAME OF PERFORMING ORGANIZATION MIT Laboratory for Computer Science / 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION Office of Naval Research/Department of Navy
6c ADDRESS (City, State, and ZIP Code) 545 Technology Square Cambridge, MA 02139	7b ADDRESS (City, State, and ZIP Code) Information Systems Program Arlington, VA 22217
8a NAME OF FUNDING/SPONSORING ORGANIZATION DARPA/DOD / 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code) 1400 Wilson Boulevard Arlington, VA 22217	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11 TITLE (Include Security Classification)
A New Architecture for Packet Switching Network Protocols

12 PERSONAL AUTHOR(S)
Zhang, Lixia

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Technical	FROM ______ TO ______	1989 August	138

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Packet-switching, statistical-multiplexing, window flow control, rate flow control queueing delay

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This dissertation presents a new architecture, the Flow Network, for packet switching network protocols. The Flow Network can provide users high quality, guaranteed service in terms of average latency and throughput. Rather than an end-point control with a stateless network model, the Flow Network design emphasizes regulation of packet traffic by the network. Rather than window flow control, the Flow Network controls the average transmission rate of individual users. Rather than relying on feedback control, the Flow Network requires users to reserve resources.

An abstract entity, a flow, is defined to represent users' data transmission requests. A flow is associated with a specific set of service requirements, which allows applications to express their requirements in a quantitative manner. This specification enables the network to check whether adequate resources are available before accepting new trans-

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL Judy Little, Publications Coordinator	22b TELEPHONE (Include Area Code) (617) 253-5894	22c OFFICE SYMBOL

DD FORM 1473, 84 MAR   83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

U.S. Government Printing Office: 1985—507-047

Unclassified

19. mission requests. It also serves as a contract between the network and the user: it is used as a measure that the network service should meet, as well as a constraint that the user's transmission behavior must adhere to.

A new channel scheduling method called the VirtualClock mechanism is developed to regulate packet flows in the Flow Network. The VirtualClock mechanism monitors the average transmission rate of each statistical data flow and provides firewalls among flows, as in a TDM system, but with the statistical multiplexing advantages of packet switching.

Simulation is used as a design aid and a verification tool throughout this research. A series of controlled simulation experiments were performed to compare the performance of the Flow Network to an existing protocol architecture, TCP/IP. The results confirm that the Flow Network is able to provide requested service quality, while in the TCP/IP network packets experience long queueing delays, network resources are unfairly allocated among users, and traffic is unstable under heavy load.

A New Architecture for Packet Switching Network Protocols

by

Lixia Zhang

Submitted to the
Department of Electrical Engineering and Computer Science
on July 17, 1989 in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Massachusetts Institute of Technology
Laboratory for Computer Science
Cambridge, Massachusetts 02139

This research was supported by the Defense Advanced Project Agency, monitored by the Office of Naval Research under Contract No. N00014-83-K-0125 and monitored by the National Aeronautics and Space Administration under Contract No. NAG2-582.

A New Architecture for Packet Switching Network Protocols

by

Lixia Zhang

Submitted to the
Department of Electrical Engineering and Computer Science
on July 17, 1989, in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Abstract

This dissertation presents a new architecture, the Flow Network, for packet switching network protocols. The Flow Network can provide users high quality, guaranteed service in terms of average latency and throughput. Rather than an end-point control with a stateless network model, the Flow Network design emphasizes regulation of packet traffic by the network. Rather than window flow control, the Flow Network controls the average transmission rate of individual users. Rather than relying on feedback control, the Flow Network requires users to reserve resources.

An abstract entity, a *flow*, is defined to represent users' data transmission requests. A flow is associated with a specific set of service requirements, which allows applications to express their requirements in a quantitative manner. This specification enables the network to check whether adequate resources are available before accepting new transmission requests. It also serves as a contract between the network and the user: it is used as a measure that the network service should meet, as well as a constraint that the user's transmission behavior must adhere to.

A new channel scheduling method called the *VirtualClock* mechanism is developed to regulate packet flows in the Flow Network. The VirtualClock mechanism monitors the average transmission rate of each statistical data flow and provides firewalls among flows, as in a TDM system, but with the statistical multiplexing advantages of packet switching.

Simulation is used as a design aid and a verification tool throughout this research. A series of controlled simulation experiments were performed to compare the performance of the Flow Network to an existing protocol architecture, TCP/IP. The results confirm that the Flow Network is able to provide requested service quality, while in the TCP/IP network packets experience long queueing delays, network resources are unfairly allocated among users, and traffic is unstable under heavy load.

Thesis Supervisor: David D. Clark
Title: Senior Research Scientist

Acknowledgments

First of all, I wish to thank Dr. David D. Clark, my thesis supervisor. He introduced me to the exciting computer networking field and kindly provided me the encouragement, guidance, and support throughout my years at MIT. I learned almost everything about packet networks from him.

I would also like to thank Professor Robert Gallager and Professor David Gifford, my thesis readers, for their technical advice and feedback, which were invaluable to my thesis. Professor Gallager read the thesis with extreme care and corrected a number of technical and grammatical errors.

I sincerely thank all members, old and new, of the Advanced Network Architecture group at MIT Laboratory for Computer Science. They create a friendly, stimulating, and enjoyable research environment. In particular, I would like to thank Larry Allen, Bob Baldwin, Noel Chiappa, and Mike Greenwald for showing me how TCP works and how IP gateways run. I would like to thank Deborah Estrin for her warm friendship which continues although she has left MIT, Radia Perlman for teaching me everything about life, and Karen Sollins for always being ready to help ever since my first day joining the group. I would also like to thank Chuck Davin, Dave Feldmeier, Eman Hashem, Andrew Heybey, and Tim Shepard for their help and friendship.

Special thanks go to Noel Chiappa, Chuck Davin, and Dave Feldmeier who read earlier drafts of the thesis and helped not only to improve its readability but also to clarify many technical issues.

Finally, I wish to thank my family. My parents have been an endless source of encouragement and support. My brother, Xiru Zhang, has shared with me difficult times as well as enjoyable camping trips. And my husband, Jiahua Ma, has supported me with unceasing love during my years struggling through graduate school.

Contents

Chapter 1

Introduction

This research is motivated by performance problems in packet switching communication networks. In a span of two decades, packet switching technology has achieved a great success. Many packet switching networks are now running around the world, delivering billions of packets every day. We have learned not only how to build large networks, but also how to connect disparate ones.

However, we have yet to learn basic principles of performance control of packet switching. In particular, despite years of research effort, the problem of network congestion has not been satisfactorily solved, and *quality-of-service (QOS)* transmission, i.e. effective mixing of various service classes, has not been truly provided.

Performance in packet switching networks is an interesting and important issue. In the future it would be highly desirable to have one network that can provide telecommunication services for all applications, instead of deploying multiple specialized networks. An integrated network should provide more efficient operation and maintenance, and should readily provide services to new applications unforeseen today. Because the packet can serve as a common building block of various types of service, packet switching is an attractive candidate for this universal communication network.

This dissertation presents a new architectural design for packet switching, the *Flow Network*, that can provide users ensured service qualities. The design focuses on three major issues: where to put the control mechanism, what kind of control mechanism to use, and whether the control should be based on reservation or feedback. As opposed to end-point control with a stateless network model, this thesis argues that the network must play an active role in traffic control, because a stateless network cannot ensure service quality. As opposed to a window control mechanism, this thesis argues that packet flows should be controlled by average transmission rate, because network resources are measured in rate. And as opposed to relying on feedback control to constantly adjust users' transmission speed, this thesis also argues that the control should be based on reservation, and use feedback control only at the reservation level. The main research results are obtained through simulation experiments. Throughout the process of design and experimentation, the goal is to learn how, and how well, the performance of a packet switching network can be controlled.

In this chapter we briefly sketch the observed performance problems in today's packet switching networks, and summarize the special features of packet switching which cause the problems. Next, a brief review of previous work is presented, followed by a thesis overview.

1.1 The Problem

1.1.1 Network Congestion

In a packet switching network, data from all sources are statistically multiplexed on shared network resources; the moment-to-moment throughput requirement often varies dramatically. Whenever a traffic surge exceeds the limit, it will congest the network, and bring about undesirable consequences: creating long transfer delays, causing packet losses, or even blocking up the network, breaking down end user connections (Figure 1-1). For example, the experience from years of running the ARPA Internet shows that congestion has been the major source of vulnerability in network service [50].

Network congestion appears to end users as poor performance. It often happens that when the network traffic becomes sluggish, say due to a momentary traffic surge, transmission delays start increasing, which in turn trigger superfluous retransmissions, making the network further loaded.[1]

Although a resource shortage contributes to the problem, in many cases the scarce resources are poorly used, because of inadequate protocol design, poor protocol implementations, or lack of data traffic control [27, 38, 49]. Under overload conditions, a well-controlled system should be able to regulate demand for the scarce resources, and isolate misbehaving users; only an

[1] Jacobson at UC Lawrence Berkeley Laboratory once measured that on average TCP data packets were retransmitted 4 to 8 times (reported in TCP/IP mail).

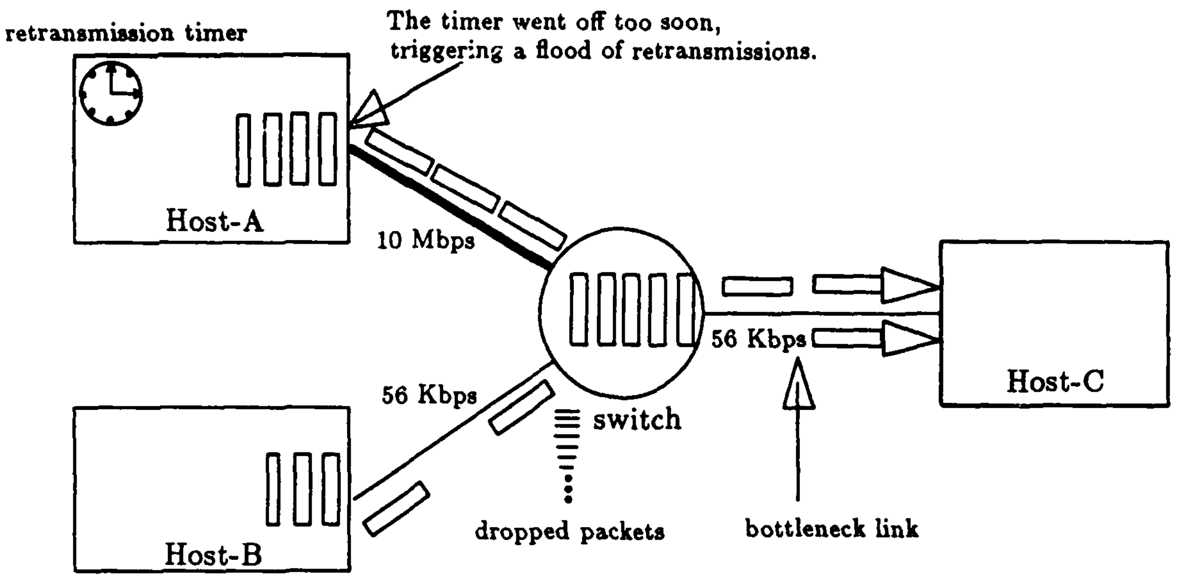


Figure 1-1: A picture of network congestion: congestion at the switch causes packet losses, while Host-A is busy with retransmissions.

uncontrolled one falls into a catastrophe.

1.1.2 Lack of Quality-Of-Service Support

As we will discuss again shortly, packet switching networks serve a broad range of applications, as well as a broad range of end equipment. The need for multiple quality-of-service (QOS) support was recognized at the very beginning: when used by different applications, the network transmission service is supposed to emphasize different performance attributes. For instance, one of the first packet switching networks, the ARPANET, was designed with special mechanisms that implicitly optimize the performance of remote login and file transfer applications.

The network architecture design, however, has fallen behind in providing effective mechanisms to support multiple QOS. The QOS selection mechanisms that currently exist in a few of the running protocols (e.g. the IP's Type-Of-Service option [40]) give only a qualitative choice, that is, a user may choose only between low or high delay, or low or high throughput classes. Undefined are the actual values implied by a choice of low or high. Furthermore, applications often desire a finer grain choice than binary on the service grade. They need a *quantitative* QOS selection.

Work on providing QOS support has been confined mainly to the area of route selection (e.g. [34]). Routing selection provides necessary, but not sufficient, QOS support. For instance, the

ARPANET is supposed to offer a short delay path between end hosts: the average transmission delay of the ARPANET was designed to be 0.2 second, which was confirmed by previous measurement under a light load [17]. Nowadays, however, due to heavy traffic load, the delay over the ARPANET is often lengthened to several seconds, which is even longer than that of satellite channels.

The above two problems, traffic congestion and lack of QOS support, reflect the same architectural defect of the current packet switching network: effective performance control components are missing from the architectural design. Congestion is merely a point in the performance spectrum indicating that the network fails to provide even the least acceptable quality of service.

1.1.3 Diverse Network Components

As shown in Figure 1-1, mismatch in network link bandwidths contributes to traffic congestion. This mismatch is the result of the way the current packet switching networks are built. During the rapid growth of computer networking, large scale networks of today have grown piecewise by connecting existing segments: local area networks are connected to each other to create larger clusters, and attach to long haul networks for a wider communication coverage. As a result, there is high heterogeneity in networking technology (LANs, satellite networks, and leased telephone wires, to list a few) and in physical characteristics of the network (transmission delays, noise, and bandwidths). Looking into a typical network, one may find switches and channel components with capacities that differ by orders of magnitude.

The heterogeneity in networking will stay with us for a long time to come, if not forever. The situation will not soon be changed by technological advances, not only because of the rapid advances of technology and the long time period needed to phase out existing systems, but also because of the *autonomy* of computer networks (i.e. users own and control their networks). Even after the last 1200 baud line is removed, the remaining network channels may still range from 1.5 Mbps to 10 Gbps or even 10 Tbps, with an even larger gap in bandwidth than what we commonly see in today's connected networks (from 1200 baud dial up channels to 10 Mbps LANs).

Some people consider the situation to somewhat resemble that of the early days of telephone networks, and expect that the standardization process will soon resolve the current problems. There exist intrinsic differences that should not be overlooked. A telephone set is of no use unless connected to the network, while all of today's LAN owners had computers long before networking them. Standardization is easier for telephone networks because of the uniform service requirements. Setting standards for networks that support diverse applications will be a different exercise. Diversities due to applications cannot be eliminated, and diversities in end machines must be accommodated.

Some people may think that the network performance problem is one of technology, rather than architecture, and that it will automatically go away with the advent of more fiber cable installations. Technology is certainly an important factor in the solution, but will not solve the problem by itself.

Technological advances indeed have brought us new communication media over the last few years. Fiber optics can now provide highly reliable and high-speed communication channels. The architectural impact of fibers is significant and yet to be fully explored. For instance, propagation delay gradually becomes a dominant factor, and millions of bytes of data can be stored in communication channels. Communication satellites are also expected to play an important role in future communication networks. Most data protocols running today, however, do not cope well with long transmission delays.

The current packet switching network already accommodates a wide spectrum of communication bandwidth. Adding fiber channels just enlarges the spectrum by another several orders of magnitude. The wider the spectrum, the more catastrophic effects network congestion may cause, once it occurs. For instance, due to signal propagation delay, a cross-country fiber channel of 1 Gbps can have 2.5 megabytes of data stored in the wire, much of which may be lost if congestion occurs on the way. Congestion control in high-speed networks poses more challenging problems than in the current networks, due to even larger speed mismatches and emergence of real-time traffic as a key application.

The heterogeneity in the packet switching network is rooted in its flexibility: packets can travel through any channel that can pass 1's and 0's. The heterogeneity is here to stay. It is the responsibility of the network architecture to accommodate the heterogeneity, smoothly gluing all the pieces together.

1.1.4 Diversity in Service Requirements

As Christian pointed out, "The difference between computers and most other machines is that computers are general purpose."[7] So are computer communication networks. Computer applications differ drastically in their required communication bandwidths, delivery delays, and reliability. When running a remote login session, most of the time the host may transmit no more than a few characters per second; when the same host starts a file transfer to ship a CAD blueprint some time later, the transmission requirement may go up to megabits per second.

Applications have different delay constraints as well. Assume that a network provides services with an average transmission delay of 0.2 second for two kinds of real-time applications, remote login, and packet voice. While the former may consider the performance satisfactory, the latter may see an excessive loss ratio, for real-time voice requires a guaranteed latency bound, rather than a low average; all overdue packets have to be discarded. At the other extreme, electronic mail applications have no concern about network delay, since, by definition, mail is not a real-time service.

Nowadays, more and more new applications are being introduced into packet switching networks, such as real-time packet voice, remote graphic application, and teleconferencing, just to name a few. These new applications demand that the network meet stringent and diverse service requirements. While conventional network applications such as file transfer require high reliability primarily and high bandwidth secondarily, real-time packet voice communication requires a very short network delivery delay as a primary consideration. Still other applications, such as digitized video, may require high throughput as the most important capability. For instance, as graphic workstations become more commonplace, scientists are developing new techniques to visualize their science [4]; a user wants to observe a smooth graphical output generated from a simulation running on a remote supercomputer. Depending on the application, delay might be secondary, and reliability of less importance [33].

Besides the application diversity, the capacity difference of end machines also contributes to diverse service requirements. Computer networks connect existing computer installations, which often happen to be a mixture of every computer system on the market, ranging from portable PCs up to giant supercomputers; the capacities differ by orders of magnitude. Consequently, their data transmission behavior differs widely, even when carrying out similar kinds of applications.

Until recently, the basic traffic in shared packet switching networks has been computer-generated data transfers from three major applications: remote login, file transfer, and electronic mail. Although each of them has different service requirements, they have tolerated low throughput and highly variant transmission delays. New applications such as real-time packet voice, however, will not work if a specified delay bound cannot be assured.

1.1.5 Packet Switching Technology

Packet switching has two distinguishing features: it permits any single user an unlimited share of the network resources (limited only by the total network capacity and the utilization by other users), and it permits an unlimited number of users to statistically share the same physical channel. A popular belief is that packet switching is feasible whenever communication bandwidths constitute the scarce resources. But bandwidth saving is only one of its advantages. This technique also provides the most flexibility among existing multiplexing techniques. Packet switching makes it possible for any user to transmit at any time and at any speed, a feature that best suits the communication needs of intelligent entities such as computers.

There is no magic, however: this flexibility brings its negative implications as well. It poses tough challenges to network control and resource management; it allows users' traffic to interfere with one another easily; it makes network load difficult both to estimate and to measure; and it can make traffic congestion too frequent.

Data packet generation in general is bursty and unpredictable [31]. Currently, there are still no basic models to describe packet generation processes or statistically multiplexed packet traffic. Nor is there a commonly agreed definition for the best measure of the performance of packet switching networks. Most commonly, average network delay, or average throughput, or power (the ratio of throughput to delay), is used in evaluating the performance of a network, but little work has been done that considers highly diverse data sources or uses multiple attributes in performance measure. And, because of the heterogeneity in applications, we speculate that a universally applicable load model may not be achievable.

The dynamic resource sharing feature, compounded with the heterogeneity in network components, end computers, and applications, makes today's packet switching networks prone to performance chaos.

In summary, the following characteristics distinguish the packet switching network from other telecommunication systems:

1. A packet switching network usually connects a diverse set of end machines.
2. A packet switching network supports a diverse set of applications. Most systems have their performance requirements specified at the system design stage, while the packet switching network must meet diverse service requests that are specified by individual applications.

3. A packet switching network may contain a diverse set of components which differ in capacity, propagation delay, and error rate characteristics.
4. Statistical resource sharing provides the needed flexibility to meet various service requirements brought forth by end diversities. The flexibility, however, can also be abused to damage the network service.

1.1.6 Summary of the Problems

It was made clear at the beginning of the packet switching era that statistical multiplexing is a feasible technology to support applications that generate data with randomness and burstiness. It has not been understood clearly, however, whether a packet switching network must include traffic control mechanisms, and if so, what kind of control mechanisms will best fit into statistical multiplexing. Especially in datagram networks, network level traffic control has been almost entirely missing.

Lack of control leaves the actual network performance to be dictated by the offered input. As a result, the performance of packet switching networks has largely been determined by traffic load. It is has been a common experience that network performance degrades sharply as the traffic volume reaches a moderate level.

New applications and new technologies give a strong impetus to the search for effective network control and resource management. The future success of packet switching technology will depend on how well the network can deploy new technologies to meet new requirements.

Below, let us first review previous work.

1.2 Previous Work

Much effort has been spent on the performance control of packet switching networks, as evidenced by the enormous amount of literature. It is impossible to treat fully all the previous work individually. Below we categorize related work, with special attention paid to the three major issues that our architectural design focuses: where to put the control mechanism, what kind of control mechanism to use, and whether the control should be based on reservation or feedback.

1.2.1 Network Architecture

There has long been a debate as to whether datagram or virtual circuit is a better choice for packet switching network architecture. The former relies on end-point control, and the latter emphasizes network control and reservation. The debate quieted down only recently, as most of the high-speed network designs for the future are clearly directed toward connection-oriented or partially connection-oriented approaches.

Datagram Approach

The datagram network offers a simple and flexible data transmission service, the so called "best effort" delivery. It delivers data packets as *independent* entities. Each packet, called a *datagram*, carries its own destination address, which is used by each switch node to dispatch it onto the next one. No information about user data flows is kept inside the network. A typical example is the IP protocol running in the ARPA Internet and elsewhere [40].

This "stateless" feature of datagram networks is considered highly desirable for several reasons [8]. The most important one is to provide robust services in the face of network component failures: a switch node crash does not lose any information concerning the data transmission; packets can be freely re-routed to get around failed switches. Another reason is to simplify switch node implementations and to facilitate interconnections among heterogeneous networks. Still another one is to offer network users the flexibility of building their own desired data transmission properties on top of provided network services.

Unfortunately, this "stateless" feature precludes the datagram network from having effective traffic control, and network congestion has been a serious problem. Although datagrams are supposed to be independent entities, in reality they interfere with one another through competing for shared resources. The simplicity of the datagram network is accompanied by service vulnerability in the presence of malfunctioning users.

The "best effort delivery" service provided by the datagram network admits no quantitative definition of the actual performance. In the Flow Network design that we will present in this thesis, the network service is clearly defined to be meeting users' demand up to the limit of resource capacity, and mechanisms are provided for each user to quantitatively specify service quality, which will then be assured by allocating adequate resources for the user.

Virtual Circuit Approach

The virtual circuit (VC) network[2] attempts to offer a reliable data delivery. Upon each user request, a logical connection is established hop-by-hop through the network. The VC network maintains the connection state at each switch node for two purposes: to check and remedy any "data integrity" damage, namely bit error, duplication or loss, and reordering, inside the network; and to control data flow on individual connections. Tymnet is a typical example of a VC network [44, 48].

Commercial networks, most of which adopted the VC approach, have enjoyed a rapid growth over the last two decades. Contrary to the datagram network's goal of offering a flexible data delivery service on a network substrate (which is possibly composed of a wide variety of communication media), most VC networks existing today, especially the commercial ones, have been built with narrower ranges of components and applications in mind. Namely, they employ low- to medium-speed telephone lines to connect packet switches, and offer only bi-directional, reliable data stream delivery service, mainly aiming at supporting remote login applications. Such a homogeneous environment makes the traffic pattern rather predictable and flow control easier.

Known VC networks have all adopted a window mechanism to control data flow on each virtual circuit. For instance Tymnet uses a hop-by-hop window; it also makes buffer reservation accordingly at each switch node along with the virtual connection setup. Consequently, VC networks are distinguished from their datagram counterpart by permitting selective flow control on individual user connections. In the Tymnet case, the buffer threshold conditions can also be propagated from a congested switch upstream to the traffic source to hold up further transmission momentarily (the so-called "back-pressure" effect).

A major benefit of the VC network, error detection and recovery, cannot be enjoyed by real-time applications because the retransmission is likely to violate the latency constraint. Likewise, back-pressure traffic control is also infeasible because of the queueing delays that may be introduced.

The Flow Network design follows the VC network's approach of setting up a logical connection (called a *flow*) hop-by-hop upon each data transmission request. Instead of using a window control mechanism, the Flow Network controls the average transmission *rate* of individual flows.

[2] Here we consider networks that build virtual connections internally, not those that only have a virtual circuit interface (such as Datapac [45], which has a virtual circuit interface but employs datagrams internally).

One of our major contributions is the design of a framework for a rate-based network control system.

1.2.2 Congestion Avoidance and Control in Datagram Networks

Since the datagram network has no traffic control mechanism built into it, a number of congestion avoidance and control mechanisms have been proposed to augment the basic network architecture. The *DECBIT* algorithm [42] and the *Slow-Start* [25] enhancement to TCP/IP represent the state of the art in congestion control over datagram networks. The use of a bit in the packet header as a feedback mechanism has been incorporated into the OSI connectionless networking protocol standard [1].

DECBIT is a binary-feedback congestion avoidance mechanism. It works in the following way: each packet carries a "congestion bit"; when the packet passes through a router, the bit will be set according to the router's state (lightly or heavily loaded). The data receiver feeds the status of the collected congestion bits back to the sender who then adjusts the flow control window according to the percentage of the bits set. Notice that although the congestion bit is set by the network, the control decision is made by individual end-to-end connections.

Slow-Start is based on a similar idea, but with different signaling and window adjustment strategies. It uses packet losses as the signal of congestion, and closes the flow control window to the minimum of one packet upon each loss, gradually reopening it later. We will discuss Slow-Start in more detail in Chapter 4, where simulation results of Slow-Start will also be presented.

This research takes a totally different approach to traffic control. Instead of a stateless network, we design an intelligent network to be in charge of traffic; instead of a window control mechanism, we use an average rate-based flow control; and instead of relying on feedback signals, we require users to make resource reservations.

1.2.3 Quality-Of-Service (QOS) Support

QOS Support over LANs

There exists a rich set of literature on supporting mixed data and packet voice traffic in a broadcast LAN environment (e.g. [13, 29, 46], to list a few). Basic approaches are assigning voice packets a higher priority over data, enforcing a fair transmission order, and limiting the access of data sources when the network becomes heavily loaded by voice users.

A broadcast LAN is a single piece of shared resource, where each user can directly observe the usage of the resource and react accordingly. In contrast, our work aims at providing QOS support over long haul networks, where the network must allocate a distributed set of resources to meet users' service requirements.

QOS Support over Long Haul Networks

The ARPANET was designed with implicit support for two QOS categories. Data traffic was assumed to come mainly from remote login applications and file transfer applications. Remote login applications are characterized by very short messages, and the ARPANET allows immediate transmission of single-packet messages without waiting for a virtual connection setup. To facilitate file transfers, the ARPANET also provides an automatic buffer reservation mechanism (Ready for Next Message) for multi-message transmission.

An explicit QOS declaration field appeared in some later network protocols, such as IP and SNA. A few bits (called QOS bits) in the packet header are reserved for indicating whether the expected throughput is high or low, whether the tolerable delay is low or high, etc. The QOS bits provide a one-way communication from the user to the network concerning the desired transmission service quality.

There is also effort to support QOS transmissions by network routing protocols. In [34], Gardner, Loobeek, and Cohn present a dynamic routing protocol which routes data packets through different paths according to the QOS requirements. The network neither allocates resources to individual users nor measures their throughput; data packets are forwarded on a First-Come-First-Serve basis.

The Flow Network provides users an interface with a quantitative service specification. A user is asked to specify quantitatively its desired average transmission rate and delay; a two-way communication channel is provided between the user and the network; if the required resources are available, the network assures the user the mutually agreed service quality.

1.2.4 Dynamic Transmission Rate Control

Mosely's Work

In [36], Mosely presented a distributed rate control algorithm. In the Mosely algorithm, each data packet carries the rate at which the user is transmitting. Each channel computes a transmission control rate based on the number of users and the current utilization (by summing up the transmission rate of all the active users), and replaces the user rate by this control rate in each passing packet if the former is higher. The control rate is carried to the receiver, which periodically sends control messages to the sender with updated control rate.

Several similarities exist between Mosely algorithm and the network control algorithm designed in this research. First, both use channel utilization as the control target. Second, both put the control mechanism in the network and use rate control.

The differences are that the Flow Network supports diverse user service requirements, and performs rate control enforcement. The Mosely algorithm does not support user specified throughput requirements, nor does it enforce the control. The Flow Network control is based on reservation and is shown to be stable by simulation. The Mosely algorithm is based on feedback control and, although the algorithm is proved analytically to converge under dynamic load, simulation tests show that the control produces long waiting queues (with lengths of hundreds of packets). In Chapter 2 we will give a brief analysis of the cause of these long queues when discussing the effect of feedback control delays.

Lambert's Work

Lambert proposed an end-point adaptive rate control algorithm for the NETBLT protocol [9], and tested the algorithm through simulation[32]. In his algorithm, each data receiver compares the sender-specified transmission rate against the measured average packet arrival rate to determine the network state and adjust the transmission rate accordingly.

When all user connections experience the same round-trip-time (RTT), the proposed algorithm provides fair service even when individual connections desire different throughput rates, where the fairness is defined in the max-min sense [18]. When connections have different RTT's, however, the service is no longer fair. The connections with longer paths have a higher data loss rate due to their slower response to network load increases; they also receive an smaller share of the channel bandwidth due to their slower response to network load decreases. The Flow

Network design lets the switches inform users of resource availability and therefore circumvents the difficulties caused by the differences in RTTs.

1.2.5 Schedule-Based Approach in Data Flow Control

In [37], Mukherji proposed a schedule-based approach to data traffic control. In his approach, channel bandwidths are divided into equal time slots which are then assigned to individual users. A user can send a packet by using its own slot, or using a slot whose designated user does not have data to send at the moment. Each user has an auxiliary flow control window which constrains the number of packets a user may send by using others' slots.

The Flow Network proposed in this research takes a similar approach of reserving resources for individual users. Instead of assigning channel slots to individual users, the Flow Network uses a *VirtualClock* mechanism to order packet transmission sequence. The VirtualClock achieves the functionality of the Mukherji algorithm, but with more flexibility in handling different channel bandwidths and different user throughput demands.

1.2.6 Leaky-Bucket Flow Control Algorithm

The Leaky-Bucket algorithm has been suggested as a flow control mechanism to be used at the network interface for high-speed networks [47]. It is similar to the traffic control algorithm proposed in this research. For a better comparison, we leave the discussion to Chapter 3, after presenting our control algorithm design.

1.2.7 Fair Queueing

Fair queueing is a simple strategy that provides all users a fair usage of network resources. Similar to the round-robin scheduling algorithm often used in operating systems, the basic idea of fair queueing is to transmit data from each user in turn. Hahne [18] and Demers et al. [2] have done extensive analysis work on fair queueing's performance.

As part of the Flow Network control, the VirtualClock mechanism performs a fair queueing function, where the fairness is defined to be assuring each user the requested average throughput rate. VirtualClock is a more general mechanism than serving users in a round-robin order: it can allocate any specified amount of resources to each user. Various queueing policies can all be implemented by a computation on the VirtualClock value.

1.2.8 Summary of Related Work

The three major design issues suggest an eight-fold design space, as shown in Figure 1-2,[3] where we put our work in context with others,

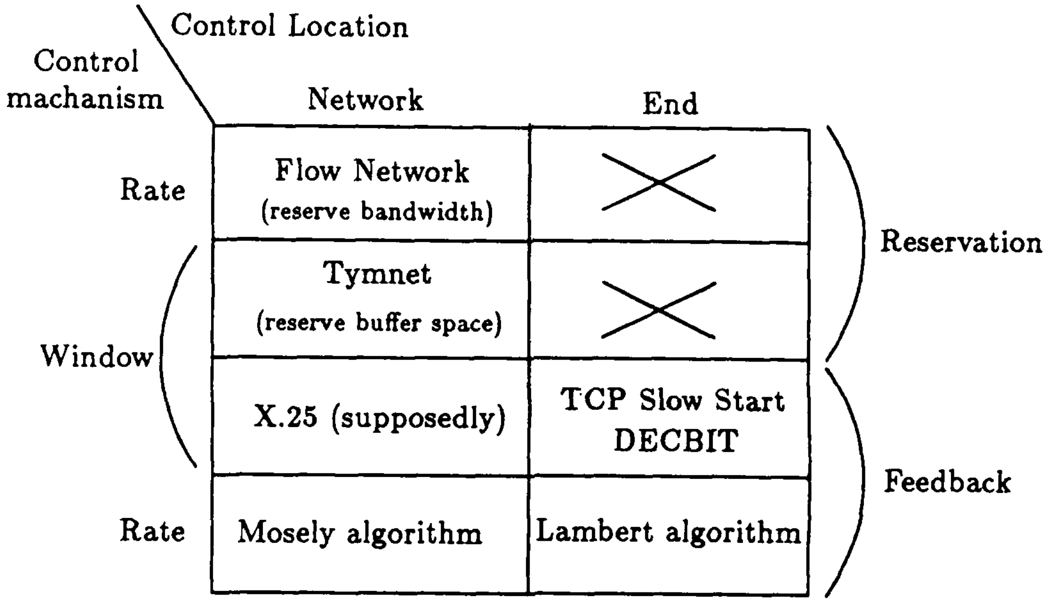


Figure 1-2: Examples of network control systems with different design decisions.

1.3 Thesis Overview

This dissertation proposes a framework of rate-based traffic control protocols, called the Flow Network, which can provide data transmissions with user-specified service quality in terms of the average throughput rate and average transmission delay. The main accomplishments are summarized below.

1. Central to the architecture is the concept of a *flow*. A flow is a stream of packets that travel through the same route from the source to the destination, and that require the same grade of transmission service. It is associated with a specific set of parameters that describes the expected transmission service.

2. An average rate control mechanism is developed.

    - Two parameters, *average rate (AR)* and *average interval (AI)*, are chosen to describe a statistical data flow.

[3]Although X.25 is only a network interface protocol, its window flow control mechanism is intended to be dynamically adjustable by feedback signals of network load.

- A variance accumulation problem in flow measurement is identified, and a user behavior envelope is proposed as the solution.
- A *simple flow monitoring and rate enforcement mechanism, VirtualClock*, is developed.

3. The design is verified and evaluated through simulation. The results confirm that a Flow Network can provide ensured average throughput rate and average transmission delay.
4. TCP/IP with Slow-Start, chosen as a representative of end control systems with a window flow mechanism, is also simulated intensively. The results show unstable network load and high queueing delays.

During the design and experiment process, we also confronted a general problem in distributed systems, control synchronization. We propose the use of randomization to suppress synchronization in distributed control.

Chapter 2 discusses our design decisions in building the Flow Network. Chapter 3 describes a realization of the new architecture, to demonstrate how the principles discussed in Chapter 2 may be applied to a practical system. It discusses problems encountered in implementing rate flow control, such as which parameters to choose as the input to the network control, and how to measure random data sources. Chapter 4 presents the experimental design and the simulation testing results. Chapter 5 concludes the work and looks into future research directions.

Most network examples and operational experience mentioned in this research are drawn from observations on the ARPA Internet, an inter-connected networking environment based in the continental US with connections to many other countries (where the ARPANET has been one of the backbone networks). Being one of the first large-scale packet switching networks, the ARPANET is described by many publications at each stage through its development. The author also has direct experience with the ARPA Internet, and a close contact with its user community. In contrast, one can seldom find published operational experience for commercial networks.

Most applications examples used in this thesis are drawn from computer communications. It should be emphasized that this research is to identify general rules of performance control, rather than proposing an architecture for specific applications (e.g. packet voice). When a packet switching network can bring the performance — mainly delay and throughput — under control, the network will be able to support various applications, including real-time voice.

1.3.1 Limitations of This Research

A simple topology model is assumed in this research. We consider a single network that consists of switch nodes and point-to-point duplex links. The network may have an arbitrary mesh topology. The links may be made of different media, such as short-wave radio, satellite channels, telephone wires, or fiber cables. The links are modeled as abstract error free communication channels with different transmission bandwidths and propagation delays.

We also assume a conventional switch model of one processing unit attached to a number of channels, and we do not consider the internal switching architecture. Switches may have different processing capacities and memory sizes, but shall all have equivalent functionalities: each can perform both as a network interface unit and as a transit node. Switches may fail; a failed switch brings down all the links attached to it.

Although there exist intrinsic relations between network routing and traffic control, these relations are not addressed by this effort. We approach the traffic control problem under the assumption that a network routing service exists which can provide routing information upon a data transmission request.

Currently there exists another obstacle to the use of packet switching for future high-speed networks: the processing overhead associated with packet switching. Rapid progress is being made in building very high-speed packet switches [22, 21]. Our research assumes that adequate processing power is available, and is concerned with how to build a packet switching network to provide ensured performance.

Chapter 2

Design Principles of the Flow Network Architecture

In this chapter, we will first specify the goals of the Flow Network architecture, and secondly, identify basic building components in it. We will then discuss the major design decisions. The chapter is outlined in Figure 2-1. A realization of the Flow Network will be presented in Chapter 3.

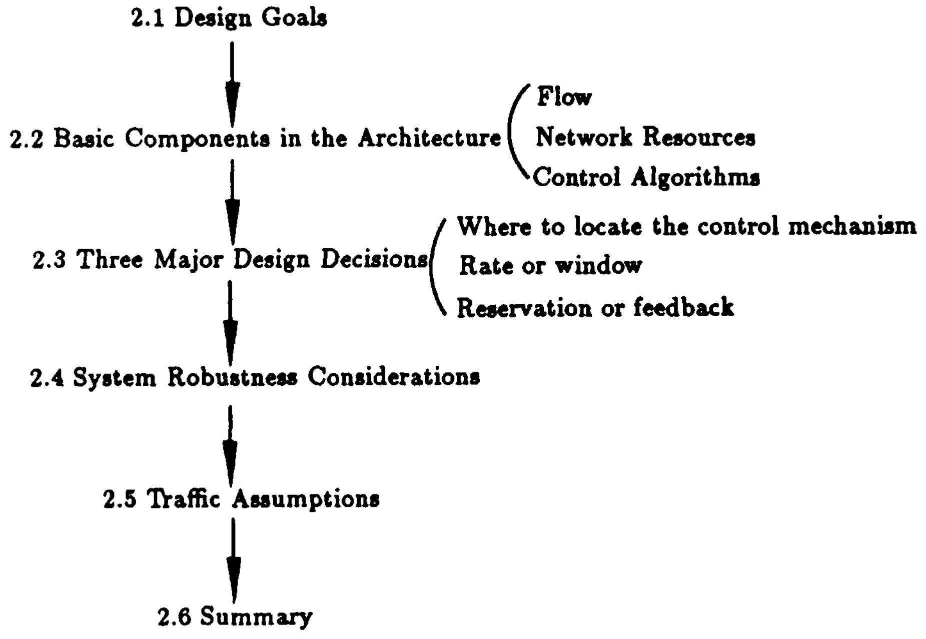

2.1 Design Goals
2.2 Basic Components in the Architecture
Flow
Network Resources
Control Algorithms
2.3 Three Major Design Decisions
Where to locate the control mechanism
Rate or window
Reservation or feedback
2.4 System Robustness Considerations
2.5 Traffic Assumptions
2.6 Summary

Figure 2-1: Chapter 2 outline.

The theme that threads through our design effort is traffic control and resource management. We identify system design issues, without worrying about implementation details, such as which functions will be implemented in hardware or which in software. The network layer is responsible for the transition from the technology dependent realm to a technology independent service for applications, but optimal implementation choices always depend on the current state of technology.

2.1 Goals of the New Architecture

We focus on the network layer in the seven-layer OSI reference model. The layers above, the end-to-end transport layer up through the application layer, represent network *users*. The word *user* in this document generally means an entity, such as a host, a process, or a human user, that requests data transmission services. Since a wide variety of users may run on a single host, each host is allowed to have a number of simultaneous data flows, each with different service requirements.

The Flow Network will offer data transmission services with the performance, in terms of average throughput rate and transmission delay, as specified by users. This implies congestion prevention. The services should be fair in terms of meeting users' throughput requests independently from the lengths of communication paths. It should also be robust, i.e., the service must withstand partial network failures.

As has been frequently observed in operational networks, users may sometimes misbehave, e.g. a user may not follow the network control protocol but rather transmit data at a high speed. Such misbehavior can be caused by software or hardware failures, by protocol implementation errors, or even by protocol design errors [38, 35]. It is the responsibility of the network control to prevent misbehaving users from interrupting normal services to others.

Above the network layer, applications generate traffic with different patterns and different service requirements. The architecture will build a service specification interface between the network and the applications. This interface will function as a set of service tuning knobs, allowing individual applications to tune the underlying network services to the desired grade.

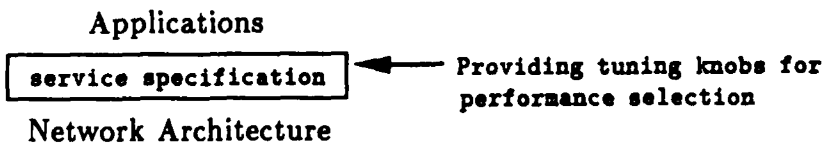


Figure 2-2: The position of the network layer.

2.2 Functional Components in the New Architecture

Stated in a simple way, a resource-sharing system generally consists of three basic types of components: the demand, the resources, and the control mechanisms that regulate the usage of resources to meet the demand. Correspondingly, the Flow Network will need representations for data transmission demands, for distributed resources, and for control mechanisms. The rest of this section discusses the first two components in more detail; the control mechanism design is the subject of the next section.

2.2.1 Data Transfer Demand: Flow

We first need a basic data transmission entity that can be used by applications to express a data transmission request, and by the network to allocate resources and to apply control actions. In datagram networks, this entity is the *packet*; in virtual circuit networks, the *virtual connection*; in the Flow Network, we define a new entity, *flow*.

A flow is a stream of packets that traverse the same route from the source to the destination, and that require the same grade of transmission service. The diversity in service requirements suggests that, together with data transmission tasks, quantitative service specifications be submitted to the network. Only with this knowledge can the control mechanism manage resources accordingly. Facing random packet traffic and unforeseen future applications, we see nothing better than a description from users as a proper estimate on their own transmission behavior.

A flow should meet the following requirements:

1. It is globally and uniquely identifiable, independent from network components.
2. It is associated with a specific set of service requirements (although the parameters may be changed during the session).
3. It is controllable, i.e. it is able to maintain the average volume of the data transmission at a specified level.

A flow differs from packets in datagram networks in that the flow is treated as one stream of packets, rather than as an aggregate of independent entities, so that the network can allocate resources to a flow, and can monitor the flow's behavior. A flow differs from a virtual connection in VC networks in that it is not concerned with data integrity. Instead, a flow is characterized by its service requirements; it is associated with the allocation and deallocation of network resources that are required to deliver the data of that flow within the specified performance bounds.

In a Flow Network, data transmissions should follow three logical phases: flow set-up, data transmission, and flow tear-down. Resources are allocated along the chosen route during the flow set-up, and deallocated by the flow tear-down. The first two logical steps may be performed simultaneously, i.e. a transmitter may start data transmission immediately following the request without waiting for the resource reservation confirmation, with the understanding that it must prepare itself for possible flow adjustment upon request, in case the network does not have adequate resources at that moment to meet the desired throughput rate.

It is up to application designers to decide when to set up a flow. A flow serves as a basic control unit to which the network both allocates resources and provides a specific grade of performance. It is not necessary to match flows with individual users or end-to-end connections. Flows can be set up when data transmission regularities can be recognized. Examples are bulk data transmissions, remote graphic applications, or a group of sessions that have the same service requirements, and that communicate with the same remote machine.

There also exist transaction-oriented applications that exchange only one or a few packets at a time, such as requests to network name servers and time servers. These *short transfer* requests do not match the *flow* model well — it is infeasible to require a flow setup or resource allocation for just a few packets. The impact and handling of short transfers will be discussed in a later section.

2.2.2 Network Resources

Network resources, i.e. the processing power, communication channels, and buffering space, are geographically distributed. They are controlled by individual switch nodes. A duplex communication link between two neighboring switch nodes is modeled as a pair of simplex channels, each belonging to the transmitting end it attaches to. With each channel is associated the information of its transmission bandwidth, delay, and error rate. A switch node should have knowledge of the capacity of its data processing, its channels, and its buffering space; all the switches together compose a loosely coupled distributed system.

Switch processors and communication channels are the driving forces in accomplishing data transmissions. We call switch processors and communication channels *data forwarding resources.* The asynchronous resource sharing feature of packet switching requires data buffering inside the network to handle momentary resource contention, but buffer space itself does not *directly* contribute to data forwarding tasks.

Data forwarding resources are measured in rate. The switch node CPU processes a certain number of packets per unit time, and the communication channels drain out a certain number of bits per unit time. The dynamic resource sharing feature distinguishes packet switching from the traditional circuit switching technology, but does not change the nature of packet switching networks as being a transmission system. In a transmission system, since the capacity is measured in rate, the resources should be allocated in rate, and the sharing among users should also be controlled in rate.

To ensure performance, each switch node must know the service requirements of individual flows in order to allocate resources accordingly. It must also monitor their resource usage. This requires that, besides the information of its forwarding resources, the switch also keep track of the service requirements and the observed behavior of individual flows (the flow state), as well as the utilization information of each piece of its resources (the switch state).

The reader may recall that one of our design goals is to provide robust services which can withstand partial network failures. It seems that, superficially, building flow state into the network may degrade its robustness. Moreover, coordination among a number of switches is also needed in order to meet flows service quality. We will discuss the system robustness issues in Section 2.4.

The following questions must be answered in switch node resources control:

1. how a switch measures its utilizations; and
2. how switches are coordinated in order to provide both good performance and robust service.

2.3 Three Major Design Decisions

This section discusses the three high-level design decisions we raised earlier (see Figure 1-2): where to put the control mechanism (inside the network or at the user end), what mechanism to use (rate or window), and what kind of control to use (reservation or feedback). We then discuss the robustness of the system. Finally, we talk about traffic assumptions.

2.3.1 Network Control versus End-Point Control

There have been two approaches to packet traffic control. One is to build a stateless network and rely on end users to adjust their transmission according to observed network state. The other approach is to build an intelligent network to control the traffic.

The stateless approach is rooted in the datagram concept. A stateless network is simple (which is an important factor in achieving high-speed) and robust (quick service recovery after a switch crash[1]). A simple architecture is always desirable, provided that it offers the needed functionality. Due to a number of limitations, however, we should not expect that a stateless network can assure high quality transmission services.

First, end-point control often needs constant feedback (e.g. acknowledgments) to determine the network state; these feedback messages increase traffic volume. In addition, end-point control, by nature, is a feedback control system, which has inherent control delay. As the network speed increases, the negative effect of a feedback control delay will also increase and the control effectiveness decrease.

Secondly, it is difficult for individual users to acquire adequate information from the feedback messages to adjust the traffic to a globally optimal value, as we see in Lambert's work.[32] Each connection can be modeled as an independent feedback control system, sharing common control signals (i.e. the network state) with all the others. Because these parallel systems have different round-trip-times (RTT's), they respond to the same input signal (i.e. the network state changes) at different speed. Fast loops respond quickly and change the system state before the information reaches longer path connections, making the latter receive out of date information.

Thirdly, a secured service quality demands secured resources, which cannot be achieved by a stateless network. There always exists a possibility of misbehaving users who do not obey the control protocol, especially in a highly heterogeneous environment. A stateless network is not able to discriminate against misbehaving users.

Recently, *Random-Drop* has been proposed as a fairness enhancement algorithm in a stateless network [39]. In essence, when a switch reaches or is about to reach a congested state, instead of discarding the next incoming packet, it drops one or more randomly picked packets from waiting queues. By probability, users who send faster than others should have more packets waiting in the queue, hence proportionally more of their packets get dropped as punishment. Simulation results, which will be presented in Chapter 5, show that Random-Drop is unable to prevent misbehaving users from taking unfair shares of network resources.

The flexibility of packet switching makes performance control more mandatory than ever. To ensure that a system will never be unusable merely due to poor performance, performance

[1]This is not necessarily true if recovery of data losses is also taken into account. A switch crash is usually accompanied by data losses.

control must be built into the network.

2.3.2 Rate versus Window Flow Control

The concept of controlling packet traffic by rate has been previously considered. In discussing network flow control, Cerf pointed out that, "It is generally the case that flow control is enforced through the allocation of permits to send packets and the reservation of buffers to receive them. ...In fact, flow control should really be dealt with by metering the rate of flow of packets into the network bound for given destinations. But for asynchronous systems, the measurement and control of rate of flow is very difficult to implement." [5] To control transmission rates of statistically varying traffic, statistics tells us to look for average properties of random events. Therefore one way to overcome the difficulty Cerf raised is to allocate resources to individual flows on an average rate basis. Controlling data traffic on time-average also implies an adequate buffer space at switch nodes to accommodate random traffic surges.

Transmission rate can provide a proper match between user demands and the resources available. A switch can determine, in terms of transmission rate, how much of a resource it can provide for a new user, or whether it can meet a requested throughput. It would be difficult to answer these questions if the switch were provided with the window sizes of flows, since in order to estimate the throughput of the new request, the switch must also know the round-trip-time of individual flows in order to convert the window size to the corresponding average throughput rate, and the information about the RTTs is often not immediately available.

Some common concerns about the feasibility of using rate flow control are that, while window flow control automatically slows data transmission if congestion occurs, rate control does not; that rate control inherently results in an unstable system, because it does not bound the amount of outstanding data; and that window control in fact paces data out by the acknowledgment returns, which are automatically adjusted to the allowed transmission rate at the bottleneck point. Below we present a brief analysis about the relative merits of window and rate flow control.

Rate or Window?

In [15] Gerla describes a so-called "self-adjustment" feature of window flow control: "If the network becomes congested, messages and acks incur high end-to-end delays. These delays, combined with the restriction on the total number of outstanding messages, effectively contribute to reduce the input rate of new packets into the network." One should realize, however, that the RTT is a random variable, causing the transmission rate to vary randomly as well; and that the throughput change always lags behind at least by one RTT time period. If at one time the RTT is lengthened by a load transient, the transmission during the next RTT period will be slowed down even if the transient is long gone.

If the heavy load persists, it implies that an excessive number of packets are *already* inside the network. Even though window flow control preserves the amount of transit data from each user, the congestion is not alleviated by any measure if the window size is not reduced. Rather, the aggressive transmission behavior, i.e. sending the next packet upon the return of each ack, keeps the network in full utilization. A fully utilized system demands a large buffer space, especially at bottleneck points.

In contrast, a network using rate control can maintain a desired utilization level by restraining flows' average transmission rate. Although rate control by itself does not limit the amount of outstanding data, combined with a proper allocation of network forwarding resources, rate control can assure that traffic input volume will not exceed the network forwarding capacity, thus avoiding data accumulation inside of the network.

One may think that the window mechanism indirectly performs rate control, in the sense that returning acks are paced out by whatever the bottleneck point along the path. This automatic pacing rate, as we have mentioned, causes the bottleneck points to run at full utilization — a very undesirable state that may result in long queueing delays.

Window flow control also introduces the overhead of acknowledgment packets, which increase the total number of packets traversing the network. In theory, a piggybacking mechanism can be used to reduce this overhead. In practice, however, most applications require only one-way transmission or two-way transmission with non-symmetric load.

Window flow control has been used effectively for many years in packet switching networks, where the throughput is low and window size small. As rapid increases in communication bandwidths admit new applications with high throughput demand, the limitations of the window mechanism start showing up.

We conclude that rate-based flow control should be our choice for the Flow Network design. A rate-based control, however, should be accompanied by resource allocation mechanisms to assure unimpeded data flow. Next, we discuss resource allocation issues.

2.3.3 Achieving Control Effectiveness by Reservation

For a statistical multiplexing system, the primary tool of performance control is to maintain a proper utilization, and, therefore, the network control must be able to regulate users' data transmissions. How should this regulation be done?

A feedback control system has control delays. In a distributed system, control delay is due to both the measurement delay and communication delay. Control effectiveness depends on the system responsiveness, as compared to individual flows' stability. With fluctuating system load, a feedback control has intrinsic limitations in its responsiveness, as shown in the analysis of the Mosely algorithm and DECBIT algorithm presented in the appendix to this chapter.

This limitation is made more severe by emerging high-speed networks. A higher speed network can suffer a larger influx of data during a given control delay period than can a lower speed network in the same amount of time. Higher bandwidths make it possible for millions of bytes of data to be stored in transmission lines, enlarging the phase-lag between data transmitters and the network measurement point. Because propagation delay cannot be reduced, the higher the network speed, the further out of phase the control can be.

We conclude that, as we enter the era of high-speed networking, it is no longer feasible to rely on *feedback signaling to adjust momentary* data transmission rates. Instead, it becomes necessary for users to inform the network of their expected throughput so that adequate resources can be *reserved* to meet the demand. By reservation we do not mean to allocate resources statically to individual users. Rather, our goal is to match the resources to the statistical average of the traffic.

The new architecture is therefore designed to be a reservation-based system. Users are required to make resource reservations, to allow the network to preserve a proper utilization level. During its life time, a flow will be provided with mechanisms to readjust the reservation parameters; nevertheless, the changes must be approved by the network first.

Feedback control should be used in regulating flow reservations, where the control can have effects that last an order of magnitude longer than the RTT. Feedback control is also needed in handling unexpected or abnormal situations, such as a mismatch between a flow's claimed throughput and the measured volume, or a momentary resource shortage during recovery from a switch failure.

Reservation Overhead Justification

A realistic control system must be both effective and efficient. One major objection to a reservation approach is the associated delay and overhead. Making flow reservations indeed involves certain overhead, as we will discuss shortly, but it does not necessarily introduce extra delay. We do not require a user to wait for confirmation before starting data transmission. When a user sends a reservation request followed by data, the request immediately informs the network what should be expected. In this way, the network gets the load change information much faster than from its own measurement; it can also send faster response to the user about whether the requested transmission can be carried out.

The main overhead in making a flow request is the resource checking at each switch along the route. Although the efficiency of a control system largely depends on the design, there exist intrinsic limits of the overhead that cannot be further reduced by the design itself. For instance, even if the per-reservation overhead is reduced to the minimum possible, the total cost will still increase linearly with the number of requests. Further reduction can be achieved only by adjusting application patterns.

For example, in the telephone network, a call setup involves a certain amount of network overhead, and hence a long conversation has less overhead than 10 short calls with the same total duration. Although in general telephone calls last long enough to make the connection setup time negligible, one always pays a fixed charge for initiating a call, independently from the duration, to reflect the system cost for connection setup. This charge policy discourages the user from initiating too many short calls. The same rule shows up in post-office service as well: all parcels bear a fixed minimum charge to cover the delivery overhead, and further weight is charged incrementally. Price policy is an effective means to influence clients' behavior. A fixed initiating charge reflects the system cost on one hand, and influences user behavior on the other.

To accommodate transaction-oriented data communications, however, we consider it necessary to provide an escape from the reservation requirement to avoid the overhead in making reservations.

Reservation and Transaction-oriented Traffic

Individual short transfers are not much concerned with throughput, but in general they all expect a low transit delay. Although transmission delay is inherently a single packet phenomenon, low delay can be met only by controlling the network utilization, which in turn requires controlling the network input.

In the Flow Network design, we decided to consider the aggregation of all short transfers as a whole, estimate the total volume, and permanently set up a special flow (called *Flow-0*) to preserve resources accordingly. This strategy will work well so long as short transfers compose only a small portion of the total traffic, and the aggregate volume remains stable. It does, however, open a possibility of unexpected load fluctuation. The network must first ensure committed performance for established flows, and may have to delay, or even discard, packets of short transfers at high points of traffic fluctuation.

It may be necessary to apply some open-loop control to short transfers to make them behave in a constrained manner, so that the disturbance they generate would be negligible. Massive buffering, although not desirable for regular flows, may be used to absorb momentary load shocks from short transfers, accompanied by feedback control on long term adjustment. These are subjects for future studies.

Finally, control is not a solution to a long term resource shortage, although an effective traffic control can provide resilience by converting the problem to one of lower availability but adequate performance, instead of a catastrophe.

Service Specification Justification

To make flow reservations, the user needs to provide a set of service quality specifications. Requiring users to specify the needed service may sound like an extra step, perhaps even a difficult one. Conventionally, it is thought that packet traffic is random; a sender may not have a quantitative estimate of its own behavior, and it merely expects the network to offer a best possible service for whatever data load it generates.

This sound observation poses a problem: the network has finite resources which are shared among multiple users. As already mentioned, the packet switching network sets no upper bound on individual users' throughput. Whenever resource contention occurs, upon what basis should it resolve the contention? Without information from the users, the network will not be able to tell whether a large packet surge is from a malfunctioning host, or whether it indeed should be expected from a fast machine.

By a different school of thought, the network could observe flows' behavior and reserve resources accordingly. Besides saving the overhead of explicit service specification, this approach also saves users the difficulty of estimating their data flow. We did not explore this option in detail, due to the following reasoning:

1. Generally speaking, users can provide a better estimate of their data flow than the network can gather from measurement.
2. Due to measurement errors and delays, the network may not be able to react to flow changes in time to make the control effective. The effectiveness of measurement-based control drops sharply with the increase in channel speed or traffic randomness/burstiness.
3. Certain performance attributes, such as delay constraints, cannot be induced from flow measurement.

Another important issue is the dependency of application protocols on the network service. All higher layer protocols are designed upon certain assumptions (considered reasonable) about the network's performance. The validity of these assumptions determines not only the performance, but sometimes also the correctness of those protocols. For instance, TCP assumes that the transmission delay will never exceed a certain time period (the current value is two minutes), and built its reliability mechanism upon this assumption. Without specifying such higher layer assumptions explicitly to the network, however, there is no reason to expect that the network will never violate them.[2]

To end the discussion on the necessity of service specification, it should be pointed out that the role of the service specification is two-fold. On the one hand, it enables the network to check whether adequate resources are available before accepting new data transmission requests. On the other hand, it also serves as a contract between the network and the user: it is used as a measure for network performance, as well as a measure of the user's own transmission behavior. Slight variations from the specified values will probably be tolerated; significant departures, however, should require either previous network permission, or timely network notification.

[2]Although IP's Time-To-Live (TTL) field can be used to limit the packet life time, TCP passes no parameters to IP to set the TTL value. In real implementations, all IP packets are set to the same TTL value, independently from the transport protocols or their assumptions.

2.4 System Robustness

Since individual components in a distributed system may fail, there seems to be a conflict between achieving service robustness and service quality, which requires keeping flow state information inside of the network. In the following we discuss how to make the Flow Network control work robustly in an unreliable environment.

State, Fate-sharing, and System Robustness The state of a system consists of variables that represent the system status. In the Flow Network architecture, the state is the status of network resources (utilization), and the status of flows (requested throughput, monitored throughput, etc.). A distributed system may become vulnerable if two or more components share some vital state information. Important issues are what state information to keep, where to keep it, and how to synchronize distributed state while minimizing the probability of failure.

A distributed system achieves robustness by automatic failure detection and service recovery. Service can be recovered if its state is not lost by the failures. Conventional wisdom says that components that share the state share their fate, and ultimate robustness is achieved when no two components share fate. This implies that each network component keeps its own state only, so that *the state is never lost unless the component itself fails.* This is called the "no-fate-sharing" principle [8].

VC networks have been considered more vulnerable than datagram networks both because the former use switch ID, concatenated with a channel ID between switches, as the virtual connection ID, and because their switches share the state of data integrity of each virtual connection. Any switch failure will call for reestablishing the connection.

The TCP/IP protocol is an example of no-fate-sharing. TCP runs on top of IP's unreliable data delivery service. It achieves robustness by keeping the connection state out of the network. Theoretically, TCP can recover from any losses due to failures in the intermediate network.

Applying the above analysis, we come to the conclusions that each switch should keep the state of its own resources and demand, and be able to forward data independently from the behavior of other switches; and that each flow should acquire its needed resources from all the switches in the path itself. A switch can lose its state only by its own failure, then the resources and the state are lost together, and there should be no interruption to the data forwarding functions of other switches.

It should be made clear that the system detects *failed components*, and recovers the *in-*

terrupted services, not the failed components (which probably will need human attention). It should also be clear that a robust system must have redundant resources to take over interrupted services from failed components.

The Flow Network protocol will have no state synchronization among switches; each switch makes independent commitments to flows. It is the flow itself that behaves as a needle that threads through all pieces of resources needed by the flow. As a side effect, the flow also indirectly synchronizes the resource allocations among switches. If one piece fails, an interrupted flow can repair itself by threading in other pieces if a local patch-up is viable, or by threading resources through a different path if a global repair is necessary, assuming that adequate routing information is provided to guide the flow's repair process.

Performance and Robustness Due to communication delays and errors, components in a distributed system cannot exchange information instantly. Hence, in general, they cannot with certainty distinguish a remote failure from poor performance in a finite time period. Therefore good performance becomes a necessary condition for achieving robustness in distributed applications. If the network ensures good performance, observed abnormal conditions can then be attributed to failures with high certainty.

In summary, a robust system is one that has redundant resources, has no state sharing among components, and prevents poor performance. Our design meets these criteria. One of the difficult tasks in a distributed system is to keep state synchronized. Our design uses data flows to synchronize the resource allocations of switches for themselves; this is both efficient and robust.

2.5 Traffic Assumption

2.5.1 Traffic Controllability

Computers served by the packet switching network differ in many ways from human users served by the telephone network. First of all, unlike the rigid requirements for real-time transmission of human voice, where long delays or bandwidth variations are unacceptable, computer generated data traffic can be designed to have, at least in most cases, the flexibility of tolerating delays and channel bandwidth adjustment. Certain applications can even accept temporary postponement of services; electronic mail and background file transfers are such examples.

Therefore it is possible for the packet switching network to regulate traffic. When a telephone network is congested, calls that fail to get through may lead to further increased call attempts, creating a positive feedback. In contrast, when congestion occurs in a packet switching network, the network can regulate (i.e. schedule) new transmission requests, or adjust the data generation rates of specific data sources.

On the other hand, although it is unacceptable to ask human users for delay tolerance or speed adjustment, they can be trusted to detect, diagnose, and properly handle abnormal situations, such as detecting a failed connection, or recovering corrupted voice. Computers, generally speaking, perform well only when being explicitly informed of what to do; they do poorly in coping with unpredicted abnormal conditions. Merely blocking traffic at the network entrance, the way many of today's network protocols do (for example, the X.25 protocol), is not an acceptable control mechanism. The network should adjust traffic volume by exchanging specific control parameters with data sources.

2.5.2 Traffic Assumptions

We measure flow duration by the network round-trip-time (RTT). Data flows (flows in short) are defined as data transfers that last at least several RTT periods. Short transfers are defined as those which transmit only a small number of packets, such that the transmission will have been finished before any control information can be fed back to the sender. This research assumes that flows make up the dominant portion of total network load.

This assumption is by no means innovative. It is an *implicit* premise of most proposed flow control mechanisms, such as in the DECBIT algorithm [42], the SNA pacing [14], or the recent TCP Slow-Start algorithm [25]. If end-to-end connections last no more than a few RTTs, none of them can have much effect. We did do one thing differently here, though, which is to state this assumption *explicitly*, to show more clearly where the limitations are, and to make users be aware of this fact.

Another pragmatic argument is that short transfers generated by remote procedure calls are not likely to contribute significant load in future high-speed networks. Assume that at a given moment a 1 Gbps channel serves ten thousand active users, and that on average a remote procedure call (RPC) generates 10 Kbits, with 70% of the RPCs within the local environment. Then $(10K \times 10K \times 0.3)$ gives 30 megabits. This says that if on average every user sends one RPC per second, the link will be 3% utilized by the RPC data. Gigabit rate channels are more likely to be utilized by continuous video data, or graphic display (as a means of coping with the vast amount of data generated by supercomputers). Both will have human users as the end receiver, which also implies a long session duration.

Traffic modeling is a task of catching moving targets. Packet switching networking is still in its infancy, and new applications are yet to be developed. How well we can guide the design of new applications now will have significant influence over data traffic patterns in the future. We should and we can guide application designs to generate predictable and controllable traffic.

2.5.3 Requirements for Applications Design

From the network control viewpoint, it is not simply the number of bits to transmit, but the unpredictability of the load, that makes maintaining good performance difficult. The basic requirement from the network control viewpoint is that applications be able to provide a quantitative description of the transmission rate expected.

Predicting the data generation rate, while it may sound novel, is in fact not new. Window flow control, the most popular mechanism in use today, provides an average throughput rate of ($Window\ Size/RTT$), which implies that the user needs to predict the data rate in order to choose a proper window size.

Past experience suggests that applications are not prepared to know or specify what service they need. It is customary for computer applications running in a virtual world provided by operating systems to be unaware of the relation between that virtual world and the reality of time and space [16]. It is infeasible for communication networks to provide a similar virtual world, because of the widely distributed nature of the resources, the inherent communication delay, the substantially larger user population, and the need to connect together different real systems.

We conclude that both application designers and network builders, whenever possible, should explore regularities in network applications, and design and implement protocols to make data sources produce predictable data flows. Given the regularities and controllabilities, the network can then collaborate on traffic control with the data sources. The better the network knows the demand, the easier and more effective the control will be, the higher the achievable performance, and the higher the achievable network resource utilization. On the other hand, the network will not be able to make any performance promises if it does not have any knowledge of the traffic.

2.6 Summary

The discussion in this chapter has laid out a foundation for the Flow Network architecture. The representation of user demand will be a *flow* which is associated with a service quality requirement. The network will enforce a *reservation-based* control. And the control target will be the *average transmission rate* of flows.

Now a good control algorithm is needed to perform three tasks:

1. resource allocation, so that a flow can get adequate resources to meet performance requirements;
2. traffic monitoring, so that resources allocated to a flow will not be taken by others; and
3. traffic adjustment, so that on one hand, the network always maintains proper utilization of resources, and on the other hand, any pending user requests will be satisfied as soon as the resources become available.

Since this is a first attempt at implementing a rate control network, there are still new and unforeseen challenges lying ahead. We will encounter and resolve some of them in the next chapter.

Appendix-2A The Mosely Algorithm Analysis

The Mosely algorithm [36] is based on feedback control, and, although the algorithm is proved analytically to converge under dynamic load, simulation tests show that under the given condition the control produces long waiting queues (with lengths of hundreds of packets). We believe the algorithm failed because of a feedback control delay which is longer than the traffic stability period.

The data generation emulates 40 pairs of human conversations, with each end sending data for an average duration of 1.2 second and then switching the direction. All end-to-end connections cross two network links. Control information update is carried in data packets to the receiver, and then fed back to data sources every 100 msec. Assuming an average queue length of 5 packets in front of each network link, the round-trip-time (RTT) is about 20 msec. Further assuming that the 40 conversations are uncorrelated, on average 10% of the conversations will have switched the direction during a 120 msec control update delay. That is, by the time a control update reached the data source, it already became obsolete.

Because of the delay, the control fails in promptly reducing the transmission rate of active users when the number of users increases, and packets get queued inside the network. Queueing delay further increases the RTT, which in turn forces the control further out of phase with the load change.

Heybey reproduced Mosely's simulation model [20]. He ran tests with the utilization control parameter changed from 80% to 60%, and observed that long queues disappeared. This is explained by observing that, although the control still falls behind the load change, the lowered utilization leaves adequate space to tolerate the mismatch between the control and the load for this particular application model. In general, the choice of utilization level depends on the dynamics of the load and the response delay of the control. In particular, the control response delay of the Mosely algorithm depends on the traffic homogeneity as well.

Appendix-2B The DECBIT Algorithm Analysis

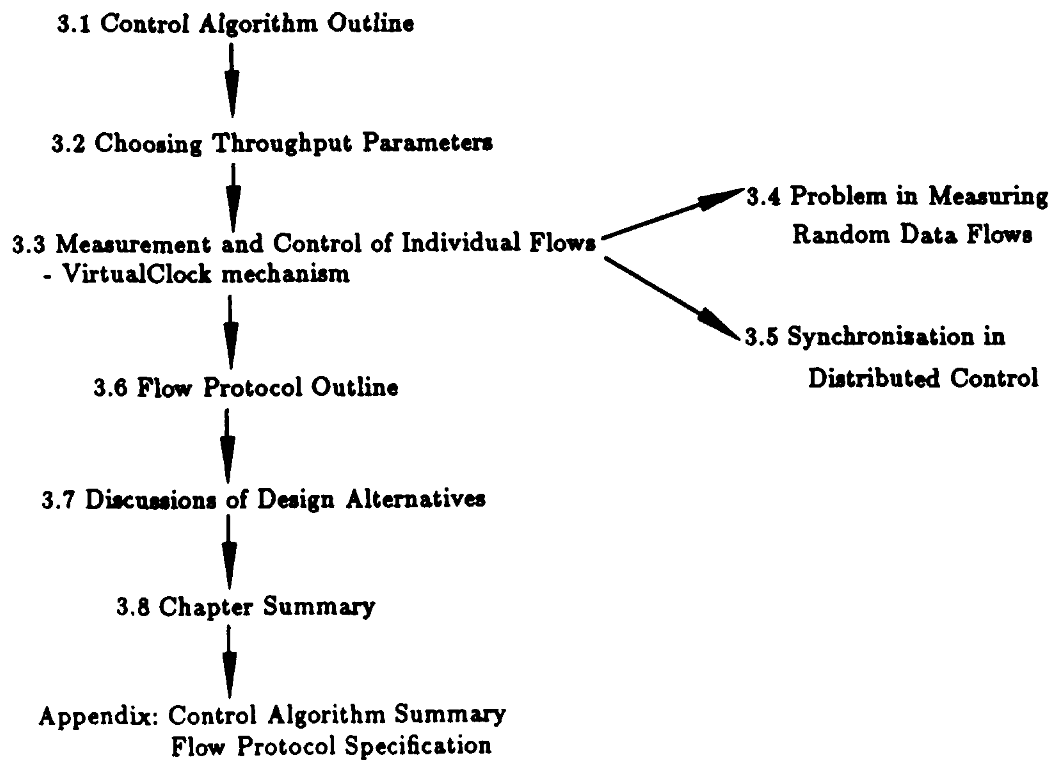
Below we consider the delay of the feedback control in the DECBIT algorithm [42, 28, 43, 6, 30]. Simulation results presented in [42] shows slow control convergence, even with a simple topology (four switches in a row) and only two users. When a second user joins the network, a rough estimate from the graph ([42], Figure 8-b) suggests that it takes more than 10 minimum round-trip-times (RTT's) (not counting queueing delay) to adjust the first user's window size by half. By computation, because the coefficient used for window adjustment is 0.875, $(0.875)^5 = 0.513$, and because after each window size change, it takes two RTTs to measure the effect, a 5-step adjustment would take a period of 10 RTTs.

Although this slow convergence is partly due to the heavy damping built into the control algorithm, it also shows the intrinsic delay of feedback control. No queueing measurement is given in the report. We speculate, however, that the control delay may well introduce very large packet queues during the transient period. Assuming that a 1-Gbps fiber channel with 10 msec propagation delay (the time to cross half of the continental US at the speed of light) meets a bottleneck point where only half of the data from this 1-Gbps channel can be passed. During a 10-RTT control adjustment period of the DECBIT scheme, probably $6 \sim 12$ megabytes of data will have to be buffered at the bottleneck point, making both the buffer space and the queueing delay a serious concern.

Chapter 3

Implementation of the Flow Network Architecture

This chapter describes one realization of the basic design principles discussed in Chapter 2. Our primary goal is to propose a framework for rate-based network control. Figure 3-1 shows an outline of the chapter.



3.1 Control Algorithm Outline
3.2 Choosing Throughput Parameters
3.3 Measurement and Control of Individual Flows - VirtualClock mechanism
3.4 Problem in Measuring Random Data Flows
3.5 Synchronisation in Distributed Control
3.6 Flow Protocol Outline
3.7 Discussions of Design Alternatives
3.8 Chapter Summary
Appendix: Control Algorithm Summary Flow Protocol Specification

Figure 3-1: Chapter 3 outline.

We start with a summary of the control algorithm, then go on to discuss problems encountered in implementing rate flow control: what parameters to choose as the input to the network control, and how to measure random data sources. During simulation tests of the proposed control algorithm, we confronted a common issue in distributed control, *synchronized control actions.* We propose the use of randomization to maintain control stability. We then present an outline of the Flow Protocol, the data carrier in the Flow Network. The chapter ends with a discussion on a few design alternatives and a comparison with the leaky-bucket mechanism. The control algorithm is summarized in Appendix-3A, and the Flow Protocol specification is given in Appendix-3B.

The design has emerged from an iterative process of simulation and contemplation. Many of the design decisions discussed in this chapter are based on simulation experience.

3.1 Traffic Control Algorithms: First Outline

The goal of network traffic control is to allocate adequate resources to individual flows to assure the service quality. As discussed in Chapter 2, one of the primary tools is the enforcement of resource reservations. To control the total traffic volume that can enter the network, all users are required to do an explicit flow set-up before or upon starting transmission.

Let us first see how the whole picture looks. Consider the network as a repository of distributed resources; a flow request collects the pieces of resources along its way. Conflicts among flows are resolved by switches, which play the role of a regulator, cutting network resources into pieces and assigning them to individual flows.

Network control performs three basic functions: decisions on accepting new flows, traffic monitoring and interference prevention among flows, and load adjustment to provide both a good service and a high utilization. Enforcing a reservation implies that new flow requests may be rejected when the network is in a resource shortage. As soon as resources become available, the network will also signal previously rejected users to resubmit the requests.

Acceptance of flow requests is decided according to the current channel utilization in order to ensure new users adequate resources. The switch estimates the current channel utilization from the sum of all the reservations made. To insure that the reservation value correctly reflects the actual channel load, each switch also constantly monitors established flows. There always exists a possibility that some flows may not follow their claimed behavior, perhaps even with justifiable reasons (e.g. an honest user may make an inaccurate transmission rate estimate). Implementation errors may also cause malfunctioning flows. Network control must be prepared to handle these possibilities. Detected mismatches between the request and the measured value will have to be corrected.[1] If a flow fails to adjust itself properly, it will be treated differently in order to prevent interference with other flows.

A mechanism called *VirtualClock* is developed both to monitor flows' behavior and to set up firewalls to prevent interference among flows. The network resource reservation control decides which flow should take what share of the resources *on average*. The *VirtualClock* determines, if more than one packet is waiting, which packet should go next based on the flows' average transmission rates.

The following issues arise as part of a rate control algorithm:

- Proper parameters are needed to specify the throughput value of a statistical flow to make resource reservation.
- The network needs a proper measurement mechanism to monitor flows.
- To behave as claimed, flow sources need a proper mechanism to control their average transmission rate.
- In performing dynamic traffic adjustment, the network must assure control stability.

In the following sections, these issues will be discussed in turn. The last one is especially interesting. One can easily list a number of examples of unstable network controls, but it is difficult to prove that a given control is stable over a wide range of operational conditions. In simulation tests we observed a potential danger of synchronized control actions, which we believe is a common issue in distributed control systems.

3.2 Choosing Throughput Parameters

To make a resource reservation, a flow sends a request message to specify the desired transmission service. We first need a set of parameters that adequately describe the behavior of a statistical data flow and can be used directly in flow measurement and control. Packet traffic is characterized as bursty and random; how should a flow describe the "burstiness" and "randomness" degree of its transmission? How should it express its throughput rate?

[1]In this research, the control mainly focuses on overflows because they may affect the overall network performance. Underflows should and can be detected by the same *VirtualClock* mechanism. However details of handling underflows are not addressed in this thesis.

Considering each flow as a statistical process, one may describe the throughput by using the average and the variance of the number of arrivals over each unit time period. Knowing the average rate helps the network control channel utilization. Measuring the average, however, is not easy; one difficulty is how to choose a proper average interval. A proper interval cannot be easily derived from the variance of arrivals over a unit time period unless a random process characterization of the data source is also known, which does not seem to be a feasible requirement. Generally speaking, an application's data generation pattern may not fit into a well known random process model, although we might be able to find a feasible model for a particular application.

Taking a simple and pragmatic approach, we choose two parameters to describe a flow's throughput: average rate (AR), and average interval (AI). That is, over each AI time period, dividing the total amount of data transmitted by the flow by AI should result in AR. Hence the variation of packet arrivals from a statistical flow is wrapped up within each average interval.

How a flow chooses its AI value is an important question. The possible range of the AI value is

$$\textit{time to send one packet} < \textit{average interval}(AI) < \textit{flow duration}$$

When the AI value reaches its lower bound, the flow would be transmitting at a constant rate as in a circuit switching network; when the AI value reaches the total duration, the flow would be able to transmit data in any arbitrary manner as in an uncontrolled packet switching network.

One may speculate that a flow would prefer an AI value as large as possible to allow itself more flexibility in data transmission. This flexibility, however, will also be accompanied with certain negative effects. First, a large AI value may reduce the responsiveness of network control, because the network averages the flow over its AI time interval before initiating any control messages to the flow source. Secondly, as will be shown later, the average interval also determines the maximum queueing delay the packet may experience at each switch; the flow has to balance the transmission flexibility with its delay tolerance. On the other hand, the network also wants to bound the AI value in order to bound the tolerance region of traffic burstiness. How the network sets this bound will be discussed shortly.

To adjust a flow's throughput in case the network cannot meet the original requested value, a third parameter is needed in a flow request in addition to AR and AI: *Lower-bound Average Rate* (LAR), to indicate a flow's minimum acceptable throughput rate.

In summary, three parameters are chosen in making a flow request,. AR, AI, and LAR. As the request message travels to the destination, the AI value may be adjusted by switches on the way. If $(AR > LAR)$, the value of AR may also be adjusted. The final values of AR and AI will be

$$\begin{aligned} AR &= \min(\textit{original value}, \min(\textit{available resources at } S_i)),\ S_i \in \textit{flow path} \\ AI &= \min(\textit{original value}, \min(\textit{AI bound set by } S_i)),\ S_i \in \textit{flow path} \end{aligned}$$

3.2.1 Network Bound on Average Interval

To determine the average interval, the network should take the flow volume into account. Flows of large volume need to be watched closely because their behavior has a large impact on the channel utilization, while a small flow can be allowed to use a much longer average interval.

Therefore, instead of bounding the average interval in terms of time for each of its channels, a switch sets an upper-bound, AIR, on the total amount of data from a single flow during an average interval:

$$AR \times AI \leq AIR$$

Now consider how to determine the AIR value for a given channel. Assuming that each switch has adequate buffering space, the AIR value should be set reasonably large to tolerate variation in flows' data transmission. The value of AIR should be proportional to the channel bandwidth, because the higher the channel bandwidth, either the higher a flow's AR value can be, or the longer the AI interval can be since more flows can be multiplexed on a big channel and the effect of individual flows variation would become relatively small. The value of AIR should also be proportional to the round-trip-time (RTT) of the network, because the RTT sets the lower-bound on network control delay.[2] Since the average interval determines the measurement delay, it should neither be too large to prolong the total control delay, nor be too short to severely constrain the burstiness of packet traffic. So for each channel, we should have

$$AIR = C \times \textit{Channel Capacity} \times RTT$$

where the coefficient C can be adjusted either to leave flows more flexibility or to further constrain transmission burstiness. If the switch buffer space is a consideration, the value of C can also be properly set to constrain the buffer consumption of each flow — over each average

[2]It takes a round-trip-time for a control command to reach the flow source and for the effect of that command to be observed by the command initiator.

interval the switch reserves bandwidth to forward AIR_i number of packets for $flow_i$, thus the flow should take no more than AIR_i packet buffers at any given time.

To simplify further discussions, we will assume that all data packets have a fixed length. Thus AR will be measured in packets per unit time, and the switch will check each flow, $flow_i$, after receiving every $AIR_i = AI_i \times AR_i$ packets from it.

3.3 Measurement and Control of Individual Flows

Given AR and AI, mechanisms are needed to measure flows based on these parameters, to enforce the transmission within the claimed average, and to set firewalls among flows to prevent mutual interference. A mechanism, called *VirtualClock*, is defined to perform these functions.

3.3.1 Virtual Clock

The idea of *VirtualClock* is inspired by the Time Division Multiplexing (TDM) system. A TDM system completely eliminates interference among users because individual user channels can transmit only during specific time slots. The capacity is wasted, however, when a slot is given to a flow that has no data to send at that moment; also the channel bandwidths are pre-fixed rather than dynamically adjustable. We want to achieve the firewalls of a TDM system as well as to maintain the flexibility of statistical multiplexing.

A TDM system is driven by a real time clock. A statistical multiplexing system may use a *VirtualClock* concept in a similar way. To make a statistical data flow resemble a TDM channel, let us imagine that arriving packets from the flow have a constant rate in a virtual time space. Instead of ticking by time, the *VirtualClock* of each flow can be advanced by the mean inter-packet gap at every packet arrival from that flow.

If a flow sends packets according to its specified average rate, its *VirtualClock* will indicate the expected arrival time of the next packet. So we can stamp the packet with the flow's *VirtualClock* value and use this stamp to order transmissions, as if the *VirtualClock* stamp were the real time slot in a TDM system. This idea can be implemented as follows (also see Figure 3-2): for $flow_i$,

1. At $flow_i$ set-up, the switch computes the value $Vtick_i = 1/AR_i$.
2. Upon the arrival of the first packet from $flow_i$, $VirtualClock_i \leftarrow$ real time.
3. Upon receiving every packet from $flow_i$, stamp the packet by the value of $VirtualClock_i$, and advance $VirtualClock_i$: $VirtualClock_i \leftarrow VirtualClock_i + Vtick_i$.

When the switch runs out of buffer space, it drops the last packet from the queue.

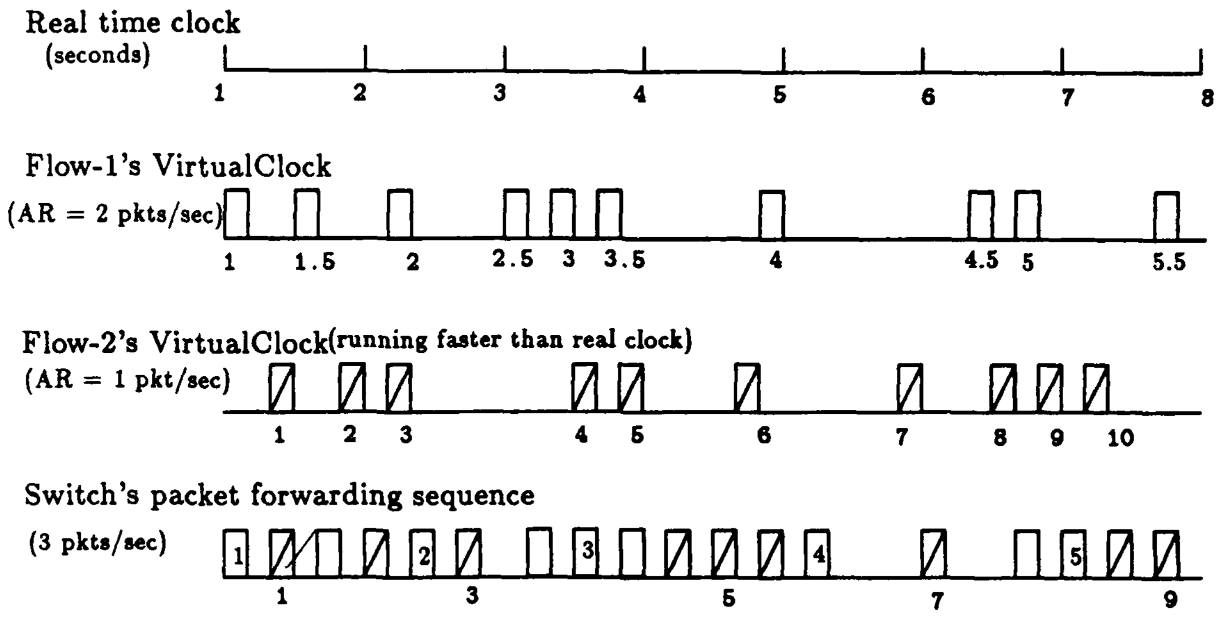


Figure 3-2: Real time, Virtual Clock, and packet processing order.

Ordering packet processing by flows' *VirtualClock* stamps assures that, although a fast running flow may take idle resources, it cannot affect other flows. In case of congestion, flows that follow their specified throughput will not be affected, while the most offending flows will receive the worst service. The *VirtualClock* mechanism precludes interference among flows, and provides performance resilience when part of the traffic runs out of control.

One major difference between a TDM system and the VirtualClock mechanism is that the latter merely *orders* packet transmission; it does not change the statistical sharing nature of packet switching. For the same reason, the *VirtualClock* mechanism can also easily accommodate priority service. Priority can be implemented by decreasing a flow's *VirtualClock* by a certain amount, P, at the start of the flow.

$$VirtualClock_i \leftarrow real\ time - P$$

where P is the priority whose value should be big enough to separate priority flows way apart from the rest in the processing queue.[3] This priority, however, does not allow the former to

[3] Using time-stamp for priority purpose has a side-effect: low priority objects will have their priority increased with time. We argue that if the channel keeps a proper utilization, P can be set to a large enough value that is longer than the resource contention period, then low priority load is effectively hidden from high priority flows. (If we define channel state from idle to next idle as an epoch, P needs be much longer than the average epoch length). Only in the presence of misbehaving users may a channel be in busy state for long, in which case we detect misbehaving users and stamp their packets with ∞.

take unfair advantage of other flows. If a prioritized flow runs too fast, its *VirtualClock* will eventually run ahead of the real time and hence its packets will lose the priority in processing.

Flow Measurement by VirtualClock

From another viewpoint, *VirtualC 'ock* plays the role of a "flow meter" that is driven by packet arrivals. The difference between the *VirtualClock* and the real time clock shows how closely the actual flow is following the specified average rate. Therefore *VirtualClock* is also used for a flow monitoring purpose.

One way to monitor flows is to let the switch check each flow's meter, *VirtualClock*, after every AI time period. But such a measure may react too slowly. A derivative detector will be able to catch misbehaving flows more quickly. That means checking the amplitude of changes.

We let the switch check each flow, $flow_i$, after receiving every $AIR_i (= AR_i \times AI_i)$ packets from it. This causes a fast flow to be watched more closely than a slow one. By counting the number of packets, traffic impulses can be detected quickly. If we had used a specific time interval for measurement, we would have faced the dilemma of picking a period that is either too small to keep control stable, or too big to detect bursts promptly.

The flow measurement can be done in the following way:

- At $flow_i$ setup, the switch computes the value $AIR_i = AR_i \times AI_i$.
- Upon receiving each set of AIR_i packets,
  - ($VirtualClock_i >$ real time) indicates that the flow has been sending faster than the specified rate. If $VirtualClock_i$ is ahead by more than a certain threshold value, control actions should be taken.
  - If ($VirtualClock_i <$ real time), then $VirtualClock_i \leftarrow$ real time.

If $flow_i$ is a prioritized one, "real time" in the above computation should be replaced by "real time − P".

We see that *VirtualClock* is driven either by incoming packets or by the real time, whichever runs faster. Credits are not saved for later use, even if the flow runs more slowly than specified. From a resource allocation viewpoint, unused resources are gone; if a flow were allowed to accumulate credits, it could increase its priority by idling for a while and then transmitting in bursts, which would cause packets from other flows to experience long queueing delays.

When the above algorithm was put into test, simulation revealed that if a burst of packets arrived from a flow that has been idle for long, the burst can still cause long queueing delays to others. This is because the flow's VirtualClock has not been advanced since the last checking point, and will not be until AIR packets are received. To solve this problem, we assign each flow an auxiliary VirtualClock (auxVC), and revise the packet stamping rule in the following way:

- At $flow_i$ set-up, the switch computes the value $Vtick_i = 1/AR_i$.
- Upon receiving the first packet from $flow_i$,
  $VirtualClock_i \leftarrow$ real time, $auxVC \leftarrow$ real time.
- Upon receiving each packet from $flow_i$,
  (1)auxVC ← max(real time, auxVC), and stamp the packet by auxVC.
  (2)auxVC ← (auxVC + Vtick), and $VirtualClock_i \leftarrow (VirtualClock_i + Vtick_i)$.

This revision replaces VirtualClock by AuxVC in packet stamping so that no flow can increase the priority of its packets by saving credits. VirtualClock remains its role as a flow meter that measures the progress of a statistic flow; its value may fall behind the real time clock between checking points to tolerate packet burstiness within an average interval.

In summary, the *VirtualClock* mechanism ensures the following properties:

- Every flow receives a fair service measured by its claimed transmission parameters.[4]
- Over-running flows can be detected by their fast running *VirtualClock*.
- Flows may be punished by longer queueing delays, or even packet losses, if they run faster than the specified rate.
- Multiple level priority services can easily be provided, and flows with priority are prevented from taking unfair advantage from others.

3.3.2 Latency Control

Because packet switching uses statistical multiplexing, data arrivals will show delay jitter to which many real-time applications are sensitive. In this section, we would like to get a first-order estimate on the queueing delay distribution and to estimate the latency bound under the VirtualClock algorithm. We conclude that increasing the channel bandwidth will be the most effective way to reduce queueing delays.

[4]The definition of fairness is a difficult subject. We consider it as a policy issue above the network control layer. The control algorithm should be able to support whatever fairness definition is given. This research assumes that the service parameters in each flow request have been checked by the fairness policy.

Queueing Delay Distribution

The end-to-end transmission delay distribution is a function of path length, channel bandwidth, channel utilization, data burstiness, and priority handling (if employed). The effect of the first two are well understood: the mean and variance of queueing delay increase with the number of switches traversed because more queues have to be crossed; the mean queueing delay is inversely proportional to the channel bandwidth (i.e. with the same packet queue distribution, the queueing delay will be decreased by a factor of N if the channel bandwidth is increased by a factor of N).

Taking bandwidths as given, we would like to get a qualitative estimate of the queueing delay distribution. Assuming packet arrivals are a random process, we use the M/M/1 queueing results in the analysis below because the distribution of an M/M/1 queue has a simple form. Although the M/M/1 queue model may not match the traffic load in real networks accurately, it is adequate for a first order approximation.

The queue length distribution for the M/M/1 queue is $P(n) = (1-\rho)\rho^n$, a geometric distribution, where ρ is the channel utilization and n the number of packets in the system (including the one under service). When $\rho \rightarrow 1$ (say 0.8 or above), or at the tail of the distribution, the probability curve of the queue length essentially follows an exponential decrease; the discrete model (geometric) approaches a continuous model (exponential) quite closely.

The variance of an exponentially distributed random variable, R, is $\sigma^2 = (E[R])^2$. Therefore packet queueing delay should be expected to have a high variance. As a first order approximation, the 99-percentile point of an exponential variable is in the vicinity of $5E[R]$.

To simplify the analysis, assuming queueing delays at each switch are independent random variables, the end-to-end delay of a flow is then

$$\sum_{i=1}^{N} D_i, \quad N = \#of\ switch\ hops\ along\ the\ path$$

In the simplest case where all channels have the same bandwidth and utilization, the sum is an Erlang distribution, with a mass distribution function of

$$1 - e^{-x/m}[\sum_{i=0}^{N-1} \frac{(x/m)^i}{i!}]$$

where m is the mean queueing delay at each queue, and N the number of queues along the path. The variance of an Erlang random variable is m^2N. The formula shows that, if m is large, it is rather difficult to achieve a tight latency bound. For example, with $N = 3$, a latency bound of about (9m + propagation delay) is needed to keep 99% of packets within the bound.

The queueing delay distribution would spread more widely with data sources that have a higher variance in packet arrivals than the Poisson model. Therefore it may become necessary to propose certain constraints on the transmission of bursty flows in order to meet latency bounds. Chapter 4 will present simulation results which show that (1) the queueing delay of Poisson data sources is smaller than that of burstier data generation models (such as the *packet train* model that will be introduced in the next chapter); and (2) proposing a proper user behavior envelope indeed helps reduce the queueing delay and the delay variation. Now we have to think of a proper way to constrain flows' transmission.

Latency Bound by Rate Control

Average-rate based traffic control is effective for controlling the average utilization. However, one difficulty in controlling a latency bound has been that average rate control does not limit the *amount* of data that may get accumulated instantly inside the network. Due to traffic burstiness, packet queues may build up momentarily even when the average utilization is low.

One way to achieve a latency bound is to enforce a strict average rate over certain time interval, i.e. to constrain a latency-sensitive flow, $flow_i$, to be sending no more than AIR_i packets per average interval.[5] Because each switch along $flow_i$'s path has committed channel bandwidths to forward AIR_i packets within each AI_i time period, no packet from $flow_i$ should wait at a switch for much longer than AI_i. To derive a worst case queueing delay bound, let N = the number of flows going through channel C, and T = the time to transmit one packet over channel C (in sec), then the bound is $W \leq AI_i + (N-1)T$ seconds.[6]

This strict rate control somewhat resembles the window flow control mechanism in constraining data transmission, except that now the "window" is opened by time rather than by

[5] As we will show in the next section, there is also another important reason for this requirement, and therefore strict rate control is required from all flows.

[6] This bound is reached only under the following extreme conditions:

1. All the auxVC's of the flows going through the same channel are behind the real time.
2. All the other flows have one or more packets arrived at the switch *instantly* and stamped by the real time value.
3. Simultaneously, all the AIR_i packets from $flow_i$ arrived.

In this case, the first packet from $flow_i$ may wait for as long as $(N-1)T$ sec, but all the AIR_i packets will be transmitted within AI_i sec after the initial wait.

acknowledgment returns. It is superior to window flow control in that, instead of waiting for acknowledgments (whose arrival time includes a random delay) before further transmission, applications can schedule ahead in data generation, and the overhead of acknowledgment is eliminated.

The Effect of Network Bandwidth

It is much easier to meet a tight latency bound with high-speed channels. Under the same packet queue distribution, queueing delay decreases inversely with channel speed. In future high-speed networks, the propagation delay may become a dominant portion of total transmission delay.

For instance, if the packet length is 250 bytes and the link propagation delay is 10 msec (half way across the continental US), then transmitting one packet over a T1 channel (1.5 Mbps) requires 0.67 msec, and emptying a queue of 20 packets (which is at the 99-percentile point of the queue length distribution of an M/M/1 queue with utilization = 80%) requires 13.4 msec, or about the same period as the propagation delay. If the channel bandwidth increases to 150 Mbps, dumping a 20 packet queue only takes 0.134 msec, which is only about 1% of the propagation delay. Although the arithmetic is simple, in simulation tests, it was still astonishing to observe how rapidly the total network transmission delay, as well as the variance of the delay, dwindles when higher speed channels are deployed.

With high bandwidth and proper rate control mechanisms, we believe that jitter in packet switching can be effectively reduced to an acceptable level for real-time applications. Chapter 4 will have further discussions on latency control issues, combined with simulation results.

3.3.3 Traffic Adjustment

There are two parameters used in the channel utilization control, $U_{capacity}$ and U_{low}. Each switch maintains the utilization of all its channels between the two boundaries (or lower than U_{low}, if the demand is low). New flow requests will be accepted only if the channel utilization can be maintained below $U_{capacity}$. When a flow terminates, the switch checks U_{low} to see if it should signal rejected requests or unsaturated flows to increase the load.

In addition, mechanisms are also needed to allow flows to adjust their volume from time to time, for example a running flow may decide to change the throughput rate. There is also a possibility of user estimation errors and therefore the specified throughput may not match the VirtualClock measured results hence corrections must be made.

Flow Initiated Adjustment

During transmission, a flow may want to change its average throughput value. The change can be carried out in the following way, similar to a flow set-up process: the flow source sends out a Flow-Inquiry message (defined in the appendix at the end of this chapter) that contains the desired change. The message then travels through each switch along the path to the destination. A decrease in the throughput rate is always approved, but a request to increase depends on the resource availability. Below we describe how a throughput increase request is carried out.

As the Flow-Inquiry message passes each switch, if any of the switches cannot approve the requested change (i.e. it has no resources to support the increase), it sets the reject bit and returns the message to the sender; otherwise it reserves the needed resources and passes the message on. When the message makes its way to the flow sink, the sink changes the message to an Inquiry-Reply and sends back to the flow source. On its way back, this Inquiry-Reply message confirms the change as it passes through each switch the second time; by this time all the switches in the path have approved the change.

As will be described in Appendix-3B, the Flow Network does not guarantee a reliable delivery of all information exchange messages between flow sources and switches (such as Flow-Inquiry). If after sending a Flow-Inquiry, a flow source receives no reply after a long wait, it may assume the message is lost and resubmit the inquiry.

Network Initiated Adjustment

Network control plays two roles in traffic adjustment: one is to correct misbehaving flows; the other is to poll waiting requests when resources become available.

Adjusting Misbehaving Flows As already mentioned, a switch can detect misbehaving flows by their fast-running VirtualClock. When a *VirtualClock* runs ahead of the real time over a certain threshold, the switch will send a control message to the corresponding flow source to demand that it slow down to the specified rate. In response, the flow should correct its transmission speed accordingly. If a higher rate is indeed needed, the flow source should go through the throughput adjustment procedure described above.

Meanwhile, the switch will accommodate the excessive data based on the resource availability. The VirtualClock mechanism assures that, in case that the reaction to the control is delayed, only the performance of the misbehaving flow may be affected. The switch should also be prepared for no response. If a flow does not respond after a number of control messages have been sent, the switch will delete the flow from its table, so that further packets from this unresponsive flow will be treated as if from an unknown user and receive the least priority in handling.

Polling Waiting Requests To serve waiting users as soon as possible, each switch keeps a record of the requests it has rejected recently due to temporary resource shortage. As soon as the resources becomes available, the switch will signal the rejected flows to re-request.

The only time a switch need check whether residual capacity exists is when a flow reduces its volume or when a flow terminates. At this time, the switch checks to see if the utilization is below U_{low}, and if so, it will first go through the list of rejected requests and send a signal to the first one or few of them (according to the available capacity). Exhausting the record for rejected flows, the switch may go to check whether any of the running flows desire a throughput increase.[7] When a flow source receives a switch signal, if it has been waiting to retry a set-up, it can then send a request. If it is running but desires a throughput increase, it should send an Flow-Inquiry message and follow the procedure described above.

The switch polls flows in turn instead of dividing the residual capacity equally among all the needed flows, because this approach has less overhead, and because if some flows can terminate as soon as possible, the rest may get an overall better service; even if the polled flow(s) lasts for long, taking flows in order still provides a sound fair service. Furthermore, flows should be satisfied most of the time so that the polling order should not be an important issue. We do not assume that flows compete for spare resources all the time.

3.3.4 Summary

This section described the proposed control algorithm for the Flow Network. Each switch adjusts the channel's utilization between two control parameters, $U_{capacity}$ and U_{low}. The *VirtualClock* mechanism plays the major role of measuring flows and setting firewalls among flows.

When putting the above control algorithm into simulation tests, we confronted two major problems, which are described in the next two sections.

[7] Whether to poll rejected or running flows first is a service policy decision.

3.4 Problems in the Behavior of Random Data Flows

3.4.1 Variance Accumulation In Statistical Flows

As proposed in the last section, the switch monitors a statistical flow by averaging its throughput after receiving each set of AIR packets. A problem left is to choose a proper threshold value, so that when ($VirtualClock-$ real time) > threshold, the switch can assume with confidence that the flow is transmitting too fast and control actions should be taken.

A number of simulation runs were conducted to test various threshold values. The results show that even when a flow generates data following an ideal Poisson process and the average interval is set to a large value (e.g. AR = 5 packets/second, AI = 10 seconds), the value of $D =$ ($VirtualClock-$ real time) may still exceed any fixed threshold after the flow is metered over a long time period. Close observations of simulation runs show that variations in a flow's data generation, surprisingly enough, have a good chance not to average out over a long period of time. The variations continuously accumulate in the *VirtualClock* measurement, and therefore the difference between the VirtualClock and the real time eventually exceeds any fixed threshold value, triggering false control actions. A simple analysis of the observed phenomenon is presented below.

First let us assume that the VirtualClock is advanced only by packet arrivals. We are interested in how the difference between a flow's VirtualClock and the real time clock may grow as time goes on.

Let us cut packet arrivals from a Poisson source into equal time intervals, $T_1, T_2, \ldots, T_i \ldots$. Letting P_i represent the number of packets arrived during T_i, we have

$$\begin{aligned} D_i &= P_i - AIR \\ Sum_n &= \sum_{i=1}^{n} D_i \\ &= (VirtualClock - RealTime)/Vtick \end{aligned}$$

The P_i's are independent, identically distributed random variables, so are the D_i's. Sum_n is a sum of n IID variables, and

$$Mean(Sum_n) = Mean(D_i) \times n = 0, \;\; Var(Sum_n) = Var(D_i) \times n \tag{3.1}$$

Equation-3.1 shows that, probabilistically, the value of Sum_n, i.e. the difference between *VirtualClock* and the real time clock, may vary above any fixed threshold, after the flow has run long enough. In fact Sum_n represents a *random walk* process, and the value of $|Sum_n|$ is unbounded as $n \rightarrow \infty$.

When $VirtualClock$ is advanced either by packet arrivals or by the real time, Sum_n in Equation-3.1 becomes

$$Sum_n = \sum_{i=1}^{n} D_i, \ \ D_i > 0 \tag{3.2}$$

Intuitively, the variance of Sum_n in Equation-3.2 should go up no slower than linearly with n. Also notice that $Var(D_i)$ is application-dependent, and so is $Var(Sum_n)$. This fact adds to the difficulty of telling whether a fast running $VirtualClock$ indicates a misbehaving flow or whether it merely indicates a large data arrival variation.

Facing this variance accumulation problem in flow measurement, a two-part solution is proposed. First and most importantly, we propose a *user behavior envelope*: we require that each flow source constrain itself from sending more than AIR packets during each average interval. As an example implementation, this user behavior envelope is enforced by a moving-average algorithm in simulation:

- The flow source keeps the transmission time of the last AIR packets in a circular ring, with a pointer, Ptr, to the oldest slot.
- When sending out a packet, P, the source checks whether the real time $\geq (Ptr(time)+AI)$. If yes, the next packet, P_{next}, can be sent after the minimum inter-packet gap; otherwise P_{next} must wait for at least an average inter-packet gap.
- P's transmission time is saved in the slot pointed by Ptr, and Ptr is moved to the next slot.

It is assumed that flows' data generators, which can be either real-time applications or data retrieval processes fetching storage, can adjust the generation rate in certain ways according to the moving-average control. Either the data rate can be adjusted without causing application performance degradation, or the data in the excessive packets (i.e. those that would have been sent without the moving-average control) can be encoded in subsequent packets.

After restricting flows within the above envelope, simulation tests show that the VirtualClock value is stabilized, varying around the real time. The control threshold is set to AIR.

At the same time, the network also makes an effort to gradually decay the variance accumulated in $VirtualClock$. When ($VirtualClock_i -$ real time) $> AIR_i$, the switch will take a control action only if the over-run is caused by a recent speed up. Otherwise it merely decays the $VirtualClock$ by a modest amount. The exact decay mechanism is described in the Flow

Monitoring section in Appendix-3A at the end of this chapter.[8]

3.4.2 Resource Overbooking

The above discussion may have triggered a related question to the reader: if the partial sum of a random data source can depart significantly from the average at a given moment, there will be flows that generate data much above the specified average, as well as flows much below the average. And for each flow, there will be periods of heavy data generation and periods of relatively low activity. Constraining a flow's transmission by a fixed envelope means cutting off the high peaks. The overall transmission rate, therefore, will be averaged lower than the specified value, and the resources may be overbooked.

Simulation tests indeed manifested such resource overbooking: when all the flows constrain their transmission according to the proposed envelope, their actual throughput is lower than the specified average (the exact value will be given in the next chapter). Enlarging the average interval can reduce, but not totally eliminate, the overbooking.

It is also possible that a user, predicting a high variation in its data generation process, may purposely specify an average rate higher than the estimated mean in order to minimize the cut-off by the rate control constraint, even if such overbooking may be associated with a higher cost.[9] Besides a reduced constraint on its data transmission, a flow that overbooks resources will also receive a better delay performance, because its *VirtualClock* will be advanced by a smaller step by each packet arrival. The performance of other running flows will not be affected by the overbooking, since *VirtualClock* assures everyone the amount of reserved resources.

3.5 Synchronization in Distributed Control

There have been many interesting stories about how seemingly random actions of independent components in a distributed system can get synchronized. As a result, the network may show oscillating behavior. We also observed synchronized control actions when testing the proposed control algorithm. This section investigates the causes of such phenomena and possible ways to reduce the synchronization.

[8]One may immediately object to this approach by saying that, "Aha, now a flow can send at a rate just a little above the specified average and will not be caught!" This is true. We do not think such small cheating can be easily distinguished from statistical fluctuations.

[9]However, the case where malicious users overbook resources to deny services to others must be prevented by proper charging or authentication mechanisms.

3.5.1 Synchronization by Common Stimuli

In simulating the proposed control algorithm, it was observed that control actions from different switches are highly synchronized. Whenever a flow increased the throughput without notifying the network first, it would receive control messages from all the switches along the way. And whenever a flow of large volume terminated, a number of switches would simultaneously discover that they now have residual capacity, and attempt to poll for new traffic at the same time.

We consider redundant control messages to be an acceptable overhead, but synchronized flow requests not. Simultaneous polls from a number of switches often result in polling in more load than the network can support at one time. Because resources are reserved when switches see a request, many requests succeed in reserving part of the needed resources before being rejected.

An example is shown in Figure 3-3, where three flow requests, F_x, F_y, and F_z, were rejected previously. When flow F_t terminated, Switches 1 ~ 3 discovered that their utilization is below U_{low} and signal the three waiting requests, respectively. But the channel between Switch-3 and Switch-4 can only afford one of the three. And in the race, the lucky flow is always F_z, since it is the closest to the bottleneck point. The resources reserved by F_x at Switch 1 and 2 are wasted for a short moment.

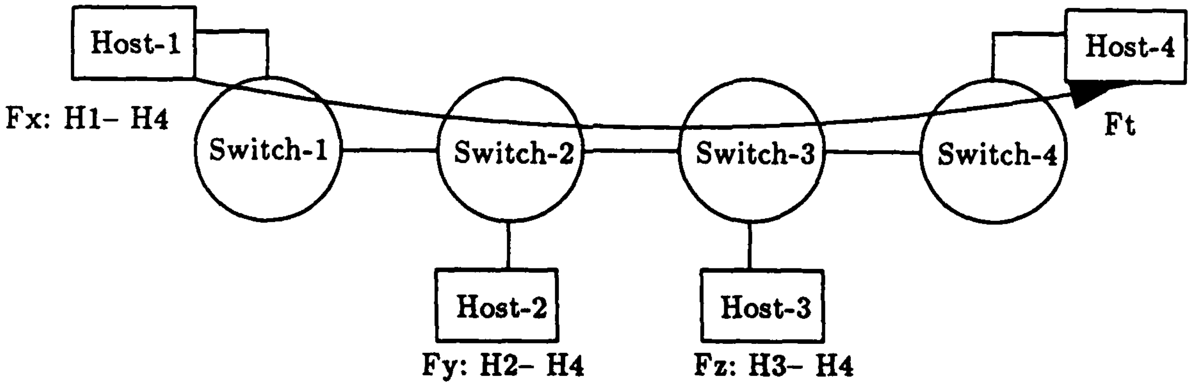


Figure 3-3: Synchronized network signaling.

A little thought reveals that this is an example of a common problem in distributed control. Another example is packet collision on an Ethernet.[10] It is observed that if a host holds the wire for a relatively long time while sending a big packet, as soon as it finishes, a number of

[10]This example is taken from the minutes of the End-to-End Task Force meeting on August 16, 1988. The End-to-End Task Force is a research group under the Internet Activity Board.

other hosts may try simultaneously to get service and collide.

Still another example is the oscillating traffic behavior observed in simulating TCP/IP protocols. Whenever a switch's packet buffer is filled up, subsequent incoming packets are dropped, and packets from a number of connections will be lost at each point of congestion. All the affected TCP connections then wait for the retransmission timeout. As the result of congestion, waits and retransmissions of multiple connections get synchronized by packet losses. Furthermore, if the connections employ the Slow-Start algorithm, the phases over which the windows shrink and reopen are highly synchronized as well. The same phenomenon is also observed independently by Hashem using a different network simulator [19]. Jacobson observed similar traffic oscillation in the ARPA Internet [24], which we believe is due to a similar cause.

A further example is the *Shortest Path First* (SPF) routing algorithm used in the ARPANET [23]. In the ARPANET, each packet-switching-node (PSN) makes *independent* routing decisions in packet forwarding. Observations showed, however, that PSNs routing decisions were highly synchronized, because each computed the routing table from the *same* network load information. As a result, traffic was switched back and forth between alternate paths (see Section 5.2 in [3] for a detailed analysis). A large damping factor has been used in the load change computation to reduce the traffic oscillation.[11]

Synchronized behavior in a distributed environment, while it may sound surprising, is easy to explain: any noticeable event occurring in the network can be observed independently by many components. If all observers, by control design, attempt the same actions accordingly, a single event will trigger synchronized reactions from all its observers. Synchronized actions are potentially detrimental to network performance. For example, in simulating the TCP/IP network, it is observed that at the peaks of load resonance, packets get dropped because of switch buffer overflow; and after the congestion the network becomes empty while the connections wait for retransmission timeout. Detailed simulation results will be presented in Chapter 4.

3.5.2 Synchronization Reduction

We propose to reduce control synchronization in the Flow Network by randomizing the timing of control actions. When a switch discovers residual capacity, it should wait for a random time period before calling for new flows. After the wait, the switch will check to confirm again the

[11] Private conversation with Marianne Lepp of BBN Communications Corporation.

availability of residual capacity before sending a poll, because other switches may have polled for new load during its wait period. The waiting time is set to be an exponential random variable, providing a degree of independence in the switch actions.

3.6 Flow Protocol

After all the previous discussion, this section presents the protocol design.

Flow Protocol (FP) is a data delivery protocol at the network layer. It provides a simplex communication channel between two end users.[12] It provides for flow clients a set of service attributes, and reserves adequate network resources to meet the requested service quality.[13]

FP consists of two modules. One module resides in host machines. It accepts data from, and delivers data to, application processes. FP transmits data with the specified average rate constraint (user behavior envelope). Periodically, the receiver sends a Flow-Status message, to be described in the appendix, to the sender to report the transmission status. The other module of FP resides in network switches. It runs the VirtualClock mechanism to order packets for processing and transmission.

All data packets carry their source and destination addresses and flow IDs. Each packet also carries an end-to-end sequence number (sort of a packet ID), which will be used in the Flow-Status message to inform the sender how well the transmission is going.

As an integral part of FP, Flow Control Protocol (FCP) behaves as an auxiliary protocol that assists FP in activating and managing flows during their active period. FCP performs flow set-up and flow tear-down, while FP carries out the data transmission phase in between. The abbreviation FCP can also be interpreted as standing for Flow Communication Protocol, since it carries out all the communications between flow users and the network, and, as a side-effect,

[12]Making FP a simplex protocol is for the sake of simplicity. FP can easily be extended to provide a duplex connection. If data from both directions travels through the same network path, a Flow-Request should contain the service parameters for the duplex channel to make the reservation. If each direction takes a different path, the flow set-up procedure needs to be changed to a three-way handshaking between the two ends A and B: when end-B receives a request from end-A, it returns an acknowledgment along the coming path, and then sends another request through a second path; when end-A receives the acknowledgment, it should wait till receiving the request from end-B, and then return an acknowledgment. The duplex flow is ready when end-B receives the acknowledgment from end-A.

[13]Transmission reliability is also one of the service parameters flow clients may choose. If the reliability option is selected, the receiver checks the integrity of data and sends negative acknowledgments, a list of missing packets, to the sender to request retransmission.

How to design a reliable protocol is not the focus in this research. We mention this point because later in simulation tests FP does check for transmission reliability, to run a fair race with TCP which is a reliable protocol.

also among switches carrying the flows. Currently nine types of FCP messages are defined, as listed in Appendix 3A at the end of this chapter.

The decision of having a separate FP and FCP is to build a simple data delivery protocol by completely separating out control messages from data packets. The early approach of carrying control information in data packet header may have bandwidth savings, but it increases processing overhead per packet. The trend of very high bandwidth optical fiber channels suggests a simple data delivery protocol with a (logically) separate control channel.

The detailed functional specification and format of FP and FCP are presented in Appendix 3A and 3B. Below we discuss two special kinds of flows, a permanent flow for short transfers, and flows that do not specify a service requirement.

3.6.1 Handling Short Transfers

The Flow Network provides a permanently established flow, named *Flow-0*, that anyone can use to send a small amount of data; the sender simply puts the source and destination addresses in the packet header, and fills the flow ID field with 0.

At each switch, Flow-0 takes a default AR value, and has a priority value of P = (−30 sec), i.e. the VirtualClock of Flow-0 is no less than 30 sec ahead of the real time. Handling of Flow-0 is the same as normal flows, except that Flow-0 cannot be throttled even if its throughput is above the default AR value. Since packets on Flow-0 receive a low priority, they are always put at the end of the transmission queue (if a queue exists), and will be dropped if the switch runs out of buffer space. As a result, applications would be discouraged from sending large quantities of data through the Flow-0 channel because of the unpredictable performance.

Unlike regular flows, the performance of Flow-0 is vulnerable to abuse. If one user sends a large amount of data through Flow-0, other Flow-0 users can be affected, because the network does not discriminate among Flow-0 users.

3.6.2 Flows without Specification

Conceivably there may exist users who do not want to specify any requirements when starting a flow. Here we propose one way to handle such *no-spec* flows. We did not test this proposal in simulation.

A no-spec flow should still send a Flow-Request before starting transmission. In the request, it can leave all the parameters unfilled (setting to value zero). After receiving the Request-Reply, the flow should transmit within the constraint of the parameter values in the reply message. It should also obey all control commands whenever it receives one.

The network may take a "best effort" attitude in providing the services. The first switch that receives such a blank request can fill the AR field with some of its residual capacity; the succeeding switches may change the AR value according to their own residual capacities. If a switch is fully utilized, it should set the Reject bit in the request and send the request back to the originator. The AI field will be filled by the switches along the way as well. In the flow table entry of a no-spec flow, the switch will set a "No-Spec" flag. It will also keep a separate count for the total load from no-spec flows.

When receiving a regular Flow Request, a switch only counts the load from regular flows to make the accept/reject decision. If deciding to accept the request, it then checks whether the total utilization, U, is below $U_{capacity}$ (by taking no-spec flows into account), and asks the no-spec flows to reduce their throughput if necessary, to maintain a proper utilization level. Therefore no-spec flows are subject to cutting down the throughput or even an interruption of service whenever the network falls short in resources. The switch does not signal interrupted no-spec flows, even when more resources become available.

The network performance perceived by such no-spec flows depends on the network load. We expect that, overall, the service would be qualitatively better that what is provided by today's datagram networks, because the flows will be informed about the network parameters and their transmission status. For instance there should be no significant data losses due to congestion, if these no-spec flows do obey the control commands.

3.7 Discussion

This section discusses several issues we encountered during the design and implementation process.

3.7.1 Phase-Locking in Rate Control

Different from the synchronized control actions we discussed earlier, the word "phase-locking" here describes a phenomenon by which packets from multiple flows always arrive at a switch simultaneously and can then only be transmitted through the same channel one by one. Consequently, packets going out last would experience a long queueing delay, and the channel resources would be utilized unevenly.

Such phase-locking occurs only by statistical chance, if the timing of rate-based data transmissions is not influenced by any network feedback signal. When data transmissions are highly regulated (e.g. sending at constant rate), however, phase-locking, once it occurs, may endure for a long time. Phase-locking has been a major concern with respect to the feasibility of rate based transmission control.

Generally speaking, applications' data generation is a random process. When certain constraints are applied to the data transmission, the randomness from the data generation phase may be reduced and regularity in the transmission increased, as is the chance of remaining phase-locked once a phase-locking occurred. With the moving-average control at flow sources, for instance, it is conceivable that, during a short period of peak data generation, transmission would pause when the quota is used up, continue again as the quota reopens with time, and may quickly pause again, waiting for the next quota reopen by time.

With high traffic multiplexing, we do not foresee phase-locking as raising serious performance problems. Because the probability of packets from a large number of independent flows arriving simultaneously is extremely low, and because there are always random factors inside the network, transient phase-locking, although it may occur, should never last for long. Chapter 4 will show simulation results that confirm our speculation.

3.7.2 Problem with Handling Priority Requests

If flow requests are allowed to have different priorities, when a high priority request enters the network, a number of switches on the path may need to cut off other low priority flows to make room for the new request. The new request therefore effectively triggers ***synchronized*** actions of the switches which may result in cutting off too much load. Consequently, the switches may soon find themselves with residual capacities and want to poll back some of the flows just stopped. During this transient period the network control itself causes some flow oscillation.

Let us use a specific example to illustrate this problem (see Figure 3-4). We assume that the capacity of all the channels is 1, $U_{capacity} = 0.8$, $U_{low} = 0.6$, Flow-1 requests a throughput of 0.1, and Flows 2 $\sim$ 5 a throughput of 0.2 each (there are other flows not shown in the figure). We also assume that the flow IDs represent the priority as well (the lower the ID number, the higher the priority).

Assume that when Flow-1's request enters the network, Flows 2 $\sim$ 5 have been running, and

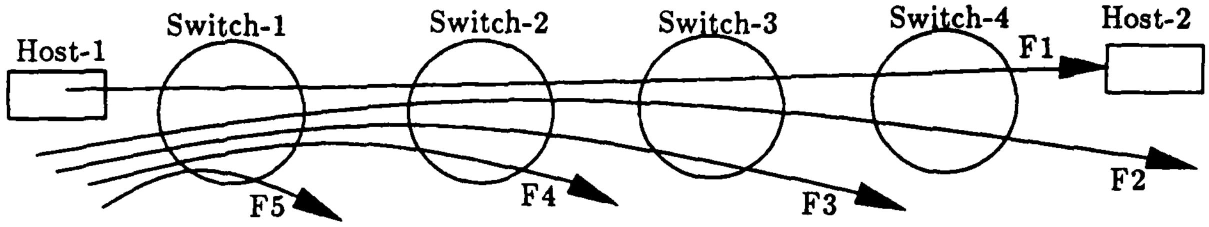


Figure 3-4: An example of transient oscillation in handling priority flows.

Switches 1 $\sim$ 4 are all running at a utilization of $U_{capacity}$. To make space for Flow-1, Switch-1 decides to terminate Flow-5 (the lowest priority flow at Switch-1); Switch-2 decides to terminate Flow-4. Switch-3 decides to terminates Flow-3, and Switch-4 terminates Flow-2. As soon as Flow-4 stops, however, Switch-1 finds that its channel utilization becomes 0.5, lower than U_{low}, and would try to poll back Flow-5.[14] Switches 2 and 3 would react similarly. Although each switch waits for a random time before sending the poll, with a 50% chance Switch-1 will poll Flow-5 before Switch-2 polls Flow-4. Assume this happens, and soon Switch-2 polls Flow-4. At this time Switch-1 has to cut off Flow-5 again to make space for Flow-4. A similar result may happen when Switch-3 polls Flow-3 after Switch-2 has re-accepted Flow-4. The worst case convergence time seems to be proportional to the network diameter.

It should be pointed out, though, that because the network control is based on reservations, the rest of the load in the network remains stable even during the transient period described above; only the affected flows may be turned on and off a few times — an effect that we consider highly undesirable. The control parameter U_{low} plays a major role here — a lower U_{low} value will make such transient unrest less likely to occur, but it may also make the resources less utilized.

Taking a conservative approach, the current design does not issue control actions that may lead to possible further load adjustment, as in the above example of handling priority requests. Dynamic load adjustment only occurs in two places. (1) When a flow wants to change its throughput, it sends an inquiry message to get network approval first. (2) When a switch discovers the channel utilization is low, it signals the sources of the rejected requests or the flows whose current rate is lower than the desired value, inviting them to re-request. Once the commitment is made, the network never revokes the resources assigned to flows except in case of hardware failures. We leave the issue of handling priority requests to future study.

[14]Switch-1 would not signal Flow-4 back because it does not know the cause of Flow-4's termination.

3.7.3 Possible Extensions to Network Control

Utilization Measurement

An important parameter in our control is $U_{capacity}$, which determines how much load each channel may get. If data sources have a high burstiness, a lower threshold may be necessary to maintain a proper queueing delay. Ideally, the value of $U_{capacity}$ should be adjusted dynamically according to traffic burstiness.

The adjustment can be based on the measurement of channel busy period length and frequency. Observations from simulation tests suggest that there is a tight correlation between the length of channel busy period and the packet queueing delay. Whenever the channel stays in the busy state for a long period, packets arriving during that period are likely to suffer a long queueing delay. Therefore we can set a threshold on the channel busy period length, and observe how frequently this threshold is exceeded. If it happens often enough, the value of $U_{capacity}$ should be reduced; if it never happens, $U_{capacity}$ can be increased. Again, reducing $U_{capacity}$ may imply cutting off running flows, an issue that requires further study.

3.7.4 Leaky-Bucket and VirtualClock

One of the flow control mechanisms often suggested for application at the user-network interface is the leaky-bucket. Various versions of the mechanism have been proposed. A simple model described in [47] works in the following way: the switch puts packets from each flow into a corresponding bucket which has a fixed size; the bucket opens periodically to drop a packet out for transmission; when the bucket is full, incoming packets are discarded. In another version of the leaky-bucket, packets from a flow are transmitted immediately if the flow has adequate transmission credits; the credits are generated at a constant rate; if no packet from the flow is sent at the moment, up to the bucket size number of credits can be saved. The difference between the two versions is that the latter allows bursty transmission.

There are similarities between Leaky-Bucket and VirtualClock. The bucket is used to smooth out variations in packet arrivals. In running VirtualClock, the average interval is the parameter that bounds the variation in data arrivals. The two also share common problems. As we have discovered, with a statistical flow, a transmission rate enforcement introduces a point of variance accumulation. If a bucket opens (or the credits are generated) at the flow's average transmission rate, it would be impossible to eliminate bucket overflow, even with large bucket sizes. Dropping packets when the bucket is full prevents the network from internal congestion, but a more satisfying solution would be to inform the user what it should do in order to avoid packet losses, i.e. to provide the user with a *behavior envelope*, such as the one suggested in this research.

One currently suggested measure to avoid the bucket overflow is to open the bucket (or to generate the credits) at a higher rate than the average transmission rate. A higher rate, however, brings up two other issues. One is that the ΔR (= bucket open rate − average transmission rate) is application dependent. To maintain a uniform loss ratio, ΔR may have to be set differently according to different variance in data generations. The other issue is that, by opening the bucket faster, the network loses track of the actual transmission rate of the user; an additional counter of some sort would be needed for measurement purpose.

A packet switch should tolerate variations in packet arrivals and should control flows' average transmission rate without sacrificing statistical multiplexing. Adjusting the bucket open rate or the credit generation rate alone does not accomplish these functions well. In the Flow Network, the problem is solved by the concept of average interval and the VirtualClock mechanism. The average interval defines checking points, and each flow wraps up variations in its transmission between two checking points. Inside the network, VirtualClock measures the average rate of flows and sets firewalls among flows at the same time. VirtualClock merely orders service and does not reduce statistical sharing. The switch forwards all packets as long as resources are available.

3.8 Summary

From the network viewpoint, transmission performance control essentially does two things: the first is resource allocation, and second usage enforcement.

This chapter presents a framework for a rate control network. The concept of an *average interval* is proposed to set checking points for measuring statistical data flows, and a *user behavior envelope* based on the average interval is proposed to restrict flows' transmission variations. The *VirtualClock* mechanism is developed as a monitoring and enforcement tool on data transmissions. Finally, randomization is used to reduce control synchronization.

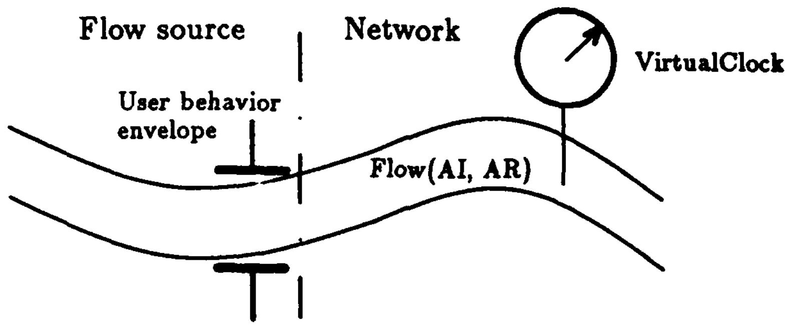


Figure 3-5: A framework for rate-based network control architecture.

Appendix-3A Summary of the Flow Network Control Algorithm

The following five procedures give a complete description of the control algorithm. The first three are about flow management: flow set-up and tear-down, flow adjustment, and network-oriented flow adjustment; the last two are about data forwarding control.

3A-1 Flow Setup and Tear-down

To set up a flow, a user fills out a Flow-Request message with the service requirements for the intended data transmission, and sends the request out. Data transmission may start immediately following the request message, if the source is prepared to recover in case that its request cannot be fully carried out. Otherwise the user should hold up data transmission until receiving a Flow-Reply to be *assured* of the requested service. To tear down a flow, either end can initiate a Flow-Terminate message. Following is a detailed description:

1. A user sends a Flow-Request message, which contains the following parameters:

AR: desired average transmission rate
AI: average interval
LAR: the lower bound of acceptable transmission rate
EXD: user expected delay
ESD: estimated delay, initialized to 0
Reject bit

The ESD field will be filled up by the switches along the way.

2. When receiving a Flow-Request, a switch first finds the corresponding out-going link, and then adds the local average delay to the ESD field in the request. If $ESD > EXD$, it sets the Reject bit in the request, turning it to a Request-Reply, and returns it to the previous switch. Otherwise the switch computes the residual capacity,

   $$\text{RC} = (\text{Link capacity } \times U_{capacity} - \textstyle\sum_{all\ passing\ flows} AR_i).$$

   If $RC < LAR$, the switch records the flow ID, the source host address, and the link ID in a "Rejected Flow" table, and rejects the request. Otherwise the switch records the flow in its flow table. If $RC < AR$, it changes the AR value in the request to RC and sets the Bottleneck flag in the entry,[15] adds the AR value into the total load, and passes the request to the next switch.

3. When a Flow-Request reaches the destination, if the destination accepts, it changes the request to a Request-Reply, keeping all the information untouched, and sends it back to the originator along the same path.

4. When receiving a Request-Reply, a switch finds the corresponding entry from its flow table. If the Reject bit is set, the switch erases the entry and releases the resources, then passes the reply to the next switch.

   If the Reject bit is off, it indicates that the set-up has succeeded. The switch compares the AR_{req} and AI_{req} values in the replay with the values in its table entry. If $AR_{req} < AR_{entry}$, then $AR_{req} \rightarrow AR_{entry}$, and the switch clears the Bottleneck flag. If $AI_{req} < AI_{entry}$, then $AI_{req} \rightarrow AI_{entry}$.

   The switch maintains in its flow table the following information about each confirmed flow:

Flow ID, to uniquely identify a flow
Source and destination addresses
Out-going link ID
Values of AR, AI, and AIR
VirtualClock, and Vtick (= 1/AR)
Status flags: No-Spec flag, Bottleneck flag
Time and type of the last control action taken on the flow

5. When the initiating user receives the Request-Reply, if the Reject bit is set, the user may try again after a random wait; the waiting time is exponentially backed-off upon each subsequent request failure. If the Reject bit is not set, all the resources along the path have been reserved and the flow enters the transmission phase.

6. To terminate a flow, either end of the flow may submit a Flow-Terminate message. Upon receiving this message, each switch releases the resources reserved for the flow and erases the flow from its flow table.

[15] Meaning that this switch is the bottleneck for this flow.

3A-2 Flow Adjustment

At any time during a flow run, the flow source may adjust the transmission rate in the following way:

1. The source sends a Flow-Inquiry message which contains the desired change value of AR, ΔAR (can be either positive or negative).
2. When receiving an inquiry message, a switch checks its resources. If it can afford the change, it changes the corresponding flow's AR to the new rate, and then forwards the inquiry to the next switch. Otherwise it changes the value of ΔAR in the inquiry to zero, the inquiry to an Inquiry-Reply, and returns it back to the sender.
3. The flow destination changes the Flow-Inquiry to an Inquiry-Reply, and sends back along the same path.
4. When receiving an Inquiry-Reply, a switch checks the ΔAR field. If $\Delta AR = 0$, the switch resets the flow's AR to the AR value in the reply (the flow's original AR value).
5. The flow source may wait for the Inquiry-Reply, or may start using the new rate right after sending the inquiry. In case the inquiry is rejected by the network, the flow source must revert the original transmission rate.

3A-3 Network Load Adjustment

When any of its flows on link i terminates or reduces the throughput, a *switch checks whether* the utilization $U < U_{low}$ for link i. If yes, the switch performs the following:

1. It looks through the "Rejected Flow" table to see whether any of the rejected requests need link i capacity. If no request is found, it looks through the flow table to find the flows with the "Bottleneck" flag set. If the switch finds a flow to take the newly available resources, it performs Step 2; otherwise it stops.
2. The switch waits for a random time period (computed from an exponential random distribution with a mean of $2 \times$ RTT[16]). After the wait the switch checks again to assure (1) it still has residual capacity and (2) the flow to be signaled has not changed. It then sends a Switch-Information message to the flow source. The Switch-Information message contains:

   The signaled flow's ID

   AR: the suggested throughput to a rejected request;

   the current throughput of a running flow

   Δ AR: possible rate increase to a running flow; not used in the other case.

[16]In the current implementation, each switch computes the network round-trip-time from the maximum ESD value it sees in all the Request-Reply messages passing by. A better implementation might be to keep the RTT value for each running flow.

3. When a rejected flow receives a Switch-Information message, it may re-submit the request immediately. When a running flow r~ceives the message, it may send a Flow-Inquiry message if it wants to increase the current throughput.

3A-4 Data Forwarding

When a switch receives a data packet, it indexes the flow table by the packet's flow ID to find the corresponding entry.

```
If VirtualClock = 0 (the first packet from that flow),
  then VirtualClock ← real time, auxVC ← real time, LastCheck ← real time;
auxVC ← max(auxVC, real time), stamp packet by auxVC, insert into a proper channel queue;
auxVC ← auxVC + Vtick, VirtualClock ← VirtualClock + Vtick;
If (VirtualClock − LastCheck) ≥ AI,
  call CheckFlow (see the next procedure).
```

When the switch runs out of packet buffers, it drops the last packet from the longest packet queue.

3A-5 Flow Monitoring

CheckFlow performs the following:

```
Let Constants: TC — time coefficient defined by the switch,
               CC — control count limit.
    Variables: $T_{last}$ — time of the last control action,
               ControlCount — number of control messages sent.
    Over ← (VirtualClock − real time)
    Period ← (real time − $T_{last}$)
If Over > AI
  if Period > TC × AI
    then VirtualClock ← VirtualClock − Over/4, $T_{last}$ ← real time;[17]
  else if Period > 2RTT
    if ControlCount > CC, erase the flow;
    else send a control message, $T_{last}$ ← real time, ControlCount ← ControlCount + 1;
else if ControlCount > 0, ControlCount ← ControlCount − 1;
```

```
If VirtualClock < real time
    VirtualClock ← real time.
```

Periodically, the switch also looks through its flow table to see if any flow has been idle for too long ((real time $-$ $VirtualClock_i$) $> TC \times AI_i$?) Flows idling for too long will be erased and their resources released.

Appendix-3B Flow Protocol Specification

In the following packet format definitions, only the content of each field, not the field length, is specified. The current specification is to be used for simulation test purposes. Detailed implementation issues, such as the address field length, the checksum algorithm, the packet sequence number wrap-around problem, protocol field for next layer, etc., have not been addressed; they must be specified before a real field trial.

3B-1 FP Header Format

1 source host address
2 destination host address
3 flow ID: 0 is reserved for short transfer packets
4 packet type: 0 specifies a data packet; all the rest is for FCP use
5 packet sequence number
6 packet length: in bytes

The header length is fixed by the protocol design. This header format is included in all FP and FCP packets.

Data Packet Format

FP packet header, type = 0
Data

3B-2 Flow Control Protocol

FCP is a "light-weight" protocol, in the sense that there is neither retransmission nor acknowl-

[1] This step decays the VirtualClock value by a moderate amount; the choice of "Over/4" is *ad hoc*.

edgment for FCP message delivery. FCP messages are *time-stamped*, rather than sequentially-numbered. A more recent message replaces early ones. All FCP messages are delivered with no further reliability enhancement above that of data packets, except that in case a switch runs out of buffer space, it should discard a data packet to make space for a received FCP packet.[18]

To reduce control delay, one special type of FCP messages, Switch-Command, is given a high priority in transmission. Switch-Command messages receive a VirtualClock stamp of (real time − 10 seconds). We try to minimize such "out-of-band" messages. The remaining FCP messages are of informative type, generated either by flow users or by switches, and are not given priority in handling. The switch stamps these FCP messages by the real time clock to ensure that they will not be lagged behind data packets from misbehaving users.

In rare cases in which control messages are not delivered in time, the control will be self-healing, in the sense that an ‹ /erload traffic condition will trigger new control messages until the reaction is observed.

Currently nine types of FCP messages are defined. Below we present the format and function of each of them.

1. Flow-Request

FP packet header, type = 1
(the part as presented in Appendix 3A-1)
N: status message parameter
Reliability flag

The handling of Flow-Request is discussed in Appendix 3A-1. The status message parameter, N, will be discussed in the Flow-Status section.

2. Flow-Request-Reply

It is exactly the same as Flow-Request, but with type = 2. The Reject bit indicates whether the request has been accepted.

3. Flow-Status

A flow sink sends a Flow-Status message after receiving every N data packets. The value of N is determined at the flow set up time. A status message contains the sequence numbers of the

[18]Under the current design, it is possible that a malicious user may flood the network with FCP messages to deny services to other users. A proposed solution is to assign each flow a dedicated FCP message buffer, so that no switch would hold more than one FCP message from the same flow.

first and last packets received (since the last Flow-Status message), stamped by the flow sink's local time.

FP packet header, type = 3	
sequence number of first packet	timestamp
sequence number of last packet	timestamp
N_{rec}: number of packets received	
negative acks (optional)	

Status messages provide a feedback channel to the flow source to allow it evaluate how well the flow is going. The flow source can collect useful information from the status reports, such as:

$$average\ packet\ arrival\ rate = \frac{last\ timestamp - first\ timestamp}{N_{rec}}$$

$$loss\ ratio = \frac{N - N_{rec}}{last\ seq\ number - first\ seq\ number}$$

If the average packet arrival rate is lower than the sending rate, it implies data accumulation inside of the network. An estimate of the RTT variance can also be computed if the sender records packet transmission times.

4. Flow-Terminate

Flow terminaticn is of the "abortive" type only, i.e. there is no graceful close. Either end of the flow may initiate a Flow-Terminate message and send to the other end. Upon seeing it, a switch erases the flow entry from its flow table.

FP packet header, type = 4

5. Flow-Inquiry

FP packet header, type = 5
AR: current average throughput rate
ΔAR: requested change

A flow may use this message to ask for throughput adjustment.

6. Inquiry-Reply

Inquiry-Reply messages have the same format as the Flow-Inquiry, with type = 6. It is converted from a Flow-Inquiry message, either by a switch or by a flow sink, by changing the type from 5 to 6. It is handled in a similar way to Request-Reply case.

7. Switch-Information

The format of Switch-Information messages is the same as the Flow-Inquiry, but with packet type = 7. It is sent from switch to flow sources to inquire a flow source whether it wants to resubmit a rejected request or to increase its transmission rate. Only switches that rejected the request, or switches that are bottleneck points of a flow, should send this inquiry when they have more resources available. Any switch on the way may stop the inquiry message (drop it), if it cannot support any more load.

Because this inquiry message is intended to be informative, no mandatory operation is defined as what a flow should do upon receiving the message. The flow source understands that the message only suggests a change rather than guarantees an offer, since other switches en route may not agree with the change, or even the originator may change its mind later.

There is no corresponding reply to Switch-Information. If the flow user does not want to make use of the information, nothing need be done. If it wants to change the average rate, it must send a Flow-Inquiry addressed to the flow sink, so that all switches en route will see the request for change. As in the Flow-Request state, the flow source may wait for a reply, or switch to the new rate immediately.

8. Switch-Command

FP packet header, type = 8
Demanded AR value
Demanded AI value

Rather than Switch-Information messages, every Switch-Command message is mandatory and must be followed. This type of FCP packets is given a high priority in delivery, because their timely delivery is important to the overall network performance, and because the network traffic is likely to be heavy when they are sent.

9. Command-Reply

The format of Command-Reply messages is the same as the command, with type = 9.

Upon receiving a Switch-Command message, the flow source should send this reply back. Command-Reply messages are assigned a normal priority in processing (stamped by the real time). Therefore, the reply would be received at about the same time by the originating switch as it should start seeing the effect of the command.

Chapter 4

Simulation Experiments and Results

In this chapter, we discuss the goals of the simulation experiments, the network model used for the test, the test plan, and then present the simulation results. Details of the simulator construction and simulation experience will not be mentioned unless they are relevant to the experimental results.

4.1 Why Simulation

Because real experiments often have limitations, the role of simulation is becoming more and more recognized as a driving function in the design and development of new systems. In our particular case, the following considerations make it infeasible to conduct real experiments to validate the Flow Network architecture in a deployed, operational network:

1. There is lack of a complete control over the experimental environment, such as load volume; it is almost impossible to make any adjustment at all when such a need occurs.
2. It is difficult to fully observe the dynamic behavior of real network operations.[1]
3. Real experiments may cause unacceptable disturbance to daily network service.

An attractive feature of simulation is its iterative nature through quick cycles. During the course of verifying the design, simulation uncovered design bugs as well as unforeseen insight.

[1] For example, the traffic oscillation behavior in the ARPA Internet was only recently observed by Jacobson [24], after a decade of operations.

The interpretation and analysis of the simulation output often stimulate further improvement to the design.

On the other hand, simulation has its own limitations. The foremost one is its accuracy in reflecting reality. Some aspects of reality, such as the traffic patterns of future applications, are even not known. The plausibility of simulation results depends on the plausibility of our modeling.

Another limitation is the computation speed. For simplicity, we built a serial event-driven simulator. It emulates a real network at packet level, and therefore simulation running time is proportional to the total number of packets transmitted. Simulation of very high-speed networks or performance estimation of very high accuracy are prohibited by the excessively long run time needed. For example, to prove that a network has a loss ratio of 10^{-9} or lower, each simulation .un should generate at least 10^{11} packets, while our simulator running on a MicroVax-3 takes about an hour to emulate 10^{7} packet transmissions. Transmitting 10^{11} packets would take 10^{4} hours, or about 420 days.

4.2 Objectives of Simulation Tests

To validate the design described in Chapter 3, we want to show that it can indeed support quantitative performance requirements. In addition, simulation should provide supporting evidence for the major design decisions made in this thesis, *reservation*-based *network* control using a *rate* mechanism.

Specifically, the simulation was designed to show the following results:

1. The Flow Network prevents congestion.
2. The Flow Network meets users' average throughput and average delay requirements.
3. The Flow Network over-performs a representative protocol architecture that relies on end control and uses a window mechanism.

4.3 Simulation Experiment Design

To plan test runs, the first step is to identify all the important factors in the experiments; and for each factor, to identify the number of values it may take.

4.3.1 Architectural Factor

As discussed in Chapter 2, the Flow Network design focused on three major issues: where to put the control mechanism, whether the control should be based on reservation or feedback, and what kind of control mechanism to use. These three issues suggest an eight-fold design space. A thorough study of these issues would be to compare the performance of all the systems with the different design decisions. It is infeasible to do so, however, in one dissertation.

We decided to evaluate the opposite corner in Figure 1-2 for the comparative study. In particular, the TCP/IP protocol suite [40, 41] is chosen for comparison with the Flow Network, and the TCP/IP implementation in Berkeley UNIX system (BSD TCP/IP), which has been widely used in the ARPA Internet and elsewhere, is chosen as the simulation model.

TCP uses an end-to-end window flow control. Most TCP implementations used a fixed window size until the last year or so, when a dynamic windc adjustment algorithm, *Slow-Start*, was developed to control congestion [25]. Even more recently, a *Random-Drop* scheme was also suggested as a congestion control strategy to assist the Slow-Start algorithm in discriminating against malfunctioning users [39].

Simulation tests have been run under two protocol architectures: the Flow Network and TCP/IP with Slow-Start enhancement. For brevity, in the rest of this document we will use the term TCP-SS to stand for the latter. A brief description of the Slow-Start algorithm is given below for readers who are not familiar with it. In simulating TCP-SS, we focus on common issues related to window mechanism and end control, rather than specific details of TCP/IP, in order to draw conclusions that are applicable to other protocols with window mechanisms and end control.

TCP/IP with Slow-Start

Because TCP runs on top of IP, a datagram protocol that provides no feedback informatioı about the network status, early implementations of TCP naturally took a fixed window size (specified by the data receiving end) as default for all connections. The idea of *Slow-Start* was first suggested by Nagle,[2] and further developed and implemented by Jacobson [25]. A dynamically adjusted congestion control window is added to TCP, which uses the acknowledgment return and the retransmission timer as input signals.

[2]in TCP/IP mail exchange.

As the name suggests, each TCP-SS connection starts with a control window size of one maximum-size packet, and opens the control window gradually upon receiving acknowledgments. The window used to regulate data transmission is min(receiver window, control window). Data losses, which are approximated by retransmission timeouts, are used as a congestion signal, upon which the control window closes down to one maximum-size packet, then reopens again as acknowledgments return. To reduce throughput losses during the period of small control window size, the window opening is divided to exponential opening phase and linear opening phase. See [25] for more details.

TCP-SS also uses a mechanism of *fast-retransmit* to quickly recover random single-packet losses. When the transmitter receives a number of acknowledgments that all acknowledge the same data sequence number, it assumes that the segment after the acknowledged number is lost, and retransmits the missing segment immediately without waiting for retransmission timeout.

TCP-SS has shown a significant performance improvement over previous TCP/IP implementations. [3]

4.3.2 Network Modeling

There are a number of factors in building a network model: topology, users' data generation model, and flow initiation model.

Network Topology

We chose a simple network topology model to be used in simulation. It has four switches in a row and connects 10 hosts (see Figure 4-2). Each link is a duplex communication channel (below we use the words link and channel interchangeably). All the links are assumed error-free. The links from hosts to the attached switch have a bandwidth of 10 Mbps, and a propagation delay of 1 msec. The three switch-to-switch links have the same bandwidth of 400 Kbps and propagation delay of 5 msec.[4] All the four switches have a moderate buffer pool size of 100 packets. The switches are assumed to have adequate capacity to process incoming packets from all attached links.

[3] A number of TCP users reported throughput improvement by a factor of 2 $\sim$ 5 in TCP/IP mail communications.

[4] The propagation speed of light on fiber is about 200,000 km/sec. A coast-to-coast span is 4000 km, with a propagation delay of 20 msec.

The path lengths of flows vary from two to four switch hops (i.e. crossing one to three switch-to-switch links). We want to see whether the path length difference, hence the RTT difference, affects users' throughput.

Data Generator Model

Data generation of a given application is an application-dependent random process. Because the packet switching network is to serve multiple current and potential applications, a universally accurate model does not exist. This is the most difficult part to model.

Most previous network performance studies, both analytical work and simulation, use the Poisson arrival model as the data generation model. There exist various speculations, however, that use of the Poisson model may not result in a realistic performance estimate. In [26], Jain and Routhier presented a *packet train* model based on their traffic measurement. We chose to use this train model for our data generation.

Modeling each packet as a railroad car, a group of packets following one another closely is modeled as a train. The generation process of a packet train model can be fully described by three parameters: train length, inter-train gap, and inter-packet gap (see Figure 4-1). Packet trains fit into a Markov chain model of two states; it is one step forward from the Poisson arrival model. Many applications can be coarsely modeled by a Markov chain (probably with more states).

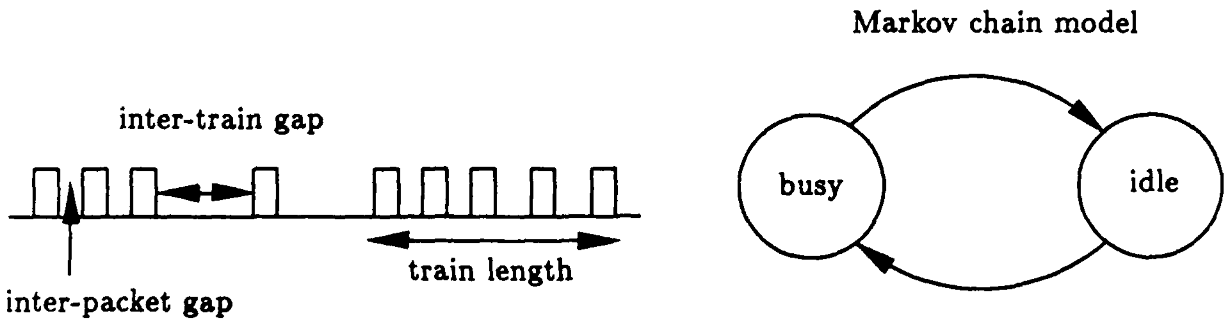


Figure 4-1: Packet train model.

Constant rate, Poisson arrival, or bursty traffic can all be considered special cases of the train model. A constant flow has the inter-packet and inter-train gap to be the same constant value. A Poisson source has a train length of one and an exponential inter-train gap. A bursty data source may have a minimal inter-packet gap.

In simulation tests, the train length is modeled as a geometrically distributed random variable. The inter-train gap is modeled as an exponentially distributed random variable. The inter-packet gap is determined by the protocols. The Flow Protocol sets the burstiness degree to 2, i.e. the inter-packet gap is $\frac{1}{2 \times Average\ Rate}$. TCP-SS uses a window flow control. When the window is open, the inter-packet gap is determined by the packet processing time;[5] otherwise the gap is lengthened by waiting for acknowledgment returns.

Several tests with Poisson data model were also conducted for comparison purpose. The results, some of which will be presented later, show that Poisson data sources lead to a smaller end-to-end delay deviation than packet train sources. Using the packet train model in testing stretches the performance of the network, demonstrating enhanced robustness of the Flow Network for a wider range of data generation patterns.

Misbehaving Data Sources

As a measure of robustness, a network control algorithm must be prepared to handle users who do not obey the control rules. We call them *misbehaving users.* This group does not include *malicious* users who attack purposely.

The simulated model of misbehaving users in the Flow Network is a data source that transmits faster than the specified rate and does not respond to network control. Such users are likely to be found in large networks.

Misbehaving users in the simulated TCP/IP network are connections that use a Go-Back-N retransmission strategy (i.e. when the retransmission timer goes off, all unacknowledged data is retransmitted) instead of executing the Slow-Start algorithm. This is a rather mild form of misbehavior. In fact, the Go-Back-N retransmission strategy may well be considered legitimate. An early TCP implementation, BSD UNIX-4.2, works exactly that way.

Connection Dynamics

Simulation tests start with a fixed set of homogeneous connections, to observe the effectiveness of rate control in a stable state. In more advanced tests, flows are also dynamically created to see how well the Flow Network performs in a dynamic environment. New transmission requests are generated as a Poisson process, and the duration is exponentially distributed. The

[5]The processing time is set to 0.1 msec in simulation.

source-destination hosts of each new flow is randomly selected from a given set of host pairs.

Because our focus is on how well the network can serve *diverse* applications, the service specification of dynamically created flows is also randomly selected from a given set. Users throughput requests are from a low of 10 Kbps to a high of 200 Kbps. We hope to gain understanding of the effects of multiplexing highly diverse data sources.

4.3.3 Performance Measure

Three basic criteria are used in the evaluation of overall performance:

1. The average throughput and delay of individual flows/connections.
2. The channel utilization and packet queue distribution at switches, to examine the system stability.
3. The fairness in service by comparing the user requested throughput with the throughput actually achieved.

In addition, the robustness of the system service is measured by how well it can keep user performance intact in the presence of misbehaving users.

4.3.4 Test Planning

In summary, the following alternatives are used in the test:

- protocol: the Flow Network, TCP-S`, TCP-SS enhanced with Random-Drop.
- data source: homogeneous, diverse, misbehaving users.
- connection: fixed, dynamic.

We divided the simulation tests into two parts. In the first part, a comparative evaluation, the following tests were performed:

1. Flow Network
   (a) Homogeneous flows.
   (b) Flows with diverse throughput rates.
   (c) The presence of misbehaving users.
2. TCP/IP with Slow-Start
   (a) Homogeneous connections.
   (b) Connections with diverse throughput rates.
   (c) The presence of misbehaving users.
3. TCP/IP with Slow-Start and Random-Drop mechanism

(a) Universally well-behaved users.
(b) The presence of misbehaving users.
(c) Starting Random-Drop before packet buffers filled.

The results from the above tests are presented in the next section.

In the second part of the simulation tests, we performed the following more advanced tests on the Flow Network.

1. Flows without the constraint of the user behavior envelope, to observe the effect of the end-point moving-average control on transmission delay.
2. Flows using different data generation models (constant rate, Poisson arrival, and packet train), to observe the relation between the data generation variation and the transmission delay variation.
3. Flows that are assigned different priorities in packet transmission, to identify the effectiveness of priority in reducing queueing delay.
4. Flows mixed with short transfers, to see how well a Flow Network can serve short transfer users.
5. Dynamically created flows, to see how well the Flow Network can handle dynamic traffic load.
6. Finally, flows running on a more complex topology (shown in Figure 4-19), to see how well the Flow Network handles the heterogeneity in links' bandwidths and delays.

The results from these tests will be presented in Section 4.5.

We did not perform confidence interval analysis on the simulation results. Our primary goal in these tests is to get a qualitative estimate about the performance of the Flow Network and to investigate the effects of various control parameters, rather than to get an accurate quantitative measure of the performance. We did run each test for a sufficiently long time and repeated each a number of times; all the runs showed consistent results.

4.4 Comparative Evaluation

Identical parameters of the data generator are used in the comparative runs. All data packets have a constant size of 250 bytes, and all FCP packets in the Flow Network and acknowledgment packets in TCP-SS have a size of 50 bytes. Network performance is measured by the channel utilization, packet queue distribution, and the fairness in service. User performance is measured by the average throughput and the end-to-end queueing delay. Because TCP-SS provides a reliable transmission service, all the flows in the Flow Network tests also have the reliability flag turned on, to provide a fair comparison.

4.4.1 Performance of the Flow Network

Flows with Same Throughput Requirement

The results presented here are from a simulation run with the following model (see Figure 4-2): there are 60 fixed flows, each generates data by the packet-train model with a mean of 10 packets/sec (20 Kbps) and requests an average throughput of 10 packets/sec. Flows 1 $\sim$ 24 have a path of 1-hop, flows 25 $\sim$ 48 2-hop, and flows 49 $\sim$ 60 3-hop.

The goal of this test is to demonstrate the performance of the Flow Network under heavy load, i.e. all the switch-to-switch channels running with the utilization = $U_{capacity}$. In this test, all the flows use the network selected bound of the average interval, 4 seconds.[6] The channel utilization control parameter $U_{capacity}$ is set to 90%. There are 18 flows on each channel. The test simulated a 10 minute run of the real system. Later we will refer to this test as Test-FP-One.

The measurement statistics of two switch channels are given below as a sample of the network performance. The channel utilization is averaged over every 100 msec period. The queue length measures the number of packets in the queue, including the one under transmission; "99-tile" means 99th percentile of the queue length samples. The effective throughput is the number of packets delivered successfully from end to end. The loss is per switch. Due to memory limitation, it is impossible to log queueing data for all the channels.

Measurement statistics with homogeneous flows

Switch	Total	Packet	Channel	Utilization		Queue Length		
ID	Forward	Loss	ID	mean	dev	mean	dev	99-tile
2	280066	0	12	0.86	0.11	2.64	1.8	10
3	280367	0	12	0.86	0.11	2.61	1.7	9

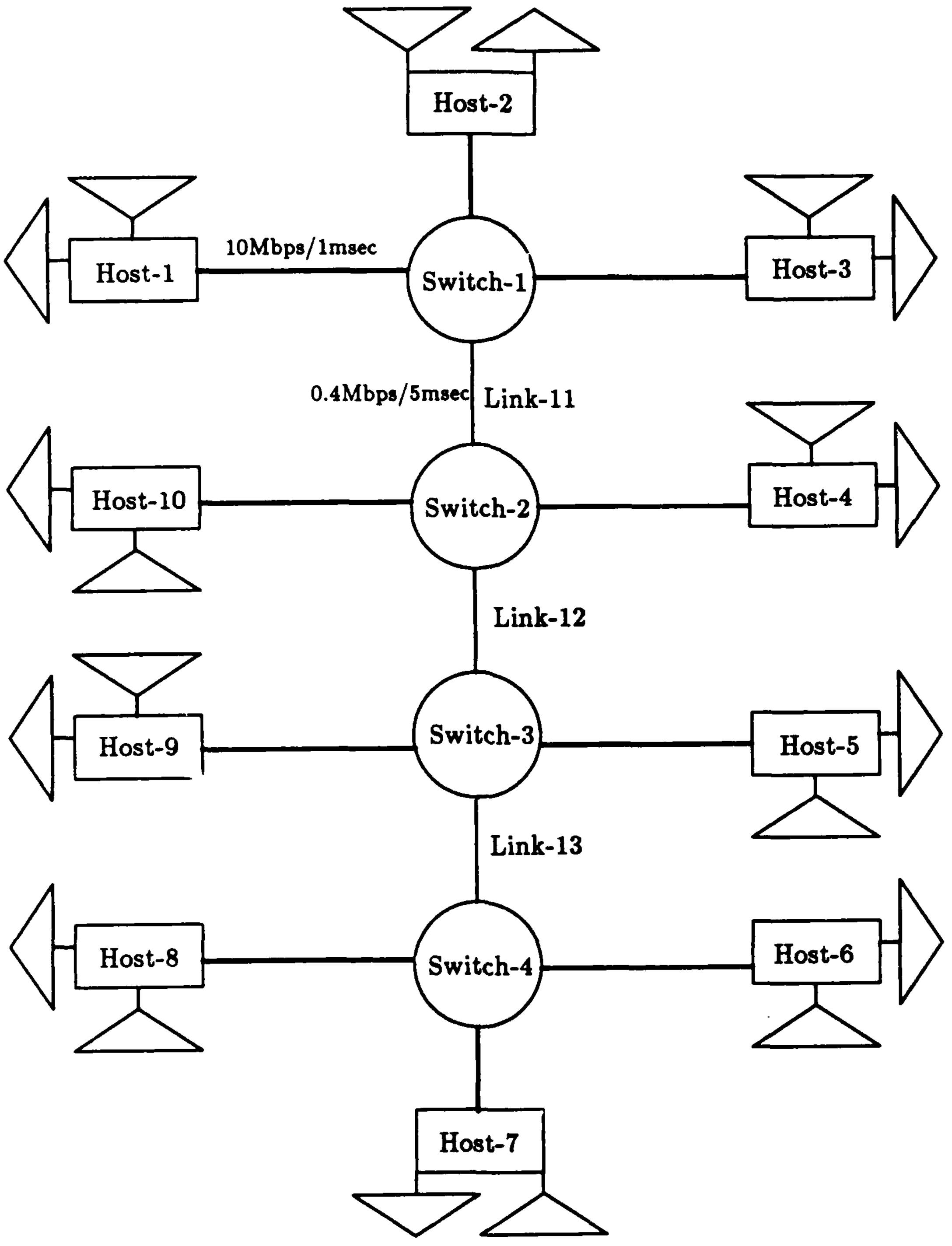
Effective throughput: 584 packets/sec

Total loss: 0

A time-history graph of the packet queue in front of Channel-12 at Switch-2 queue over 1 minute running time is also attached (Figure 4-3).

The average throughput and end-to-end queueing delay of the 60 flows are shown in Figure 4-4. Dividing the 60 flows into three path-length groups, we also computed the average throughput

[6]AIR = (C × Channel bandwidth × RTT). Here the bandwidth is 400 Kbps, a loose RTT estimate is set to 0.2 second, and C is set to 1, so AIR = 80 Kbits = 40 packets, and AI = AIR / AR = 4 seconds. I have done tests with smaller AI values (1 sec and 2 sec) which result in shorter queues at switches and lower end-to-end delays, but also a lower average throughput.



Packet size = 250 bytes

Flow average throughput = 20 Kbps

Flow average interval = 4 sec

Average packet train length = 5

Switch buffer size: 100

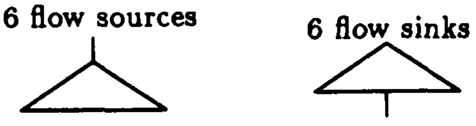


Figure 4-2: Topology and connections used for comparative simulation.

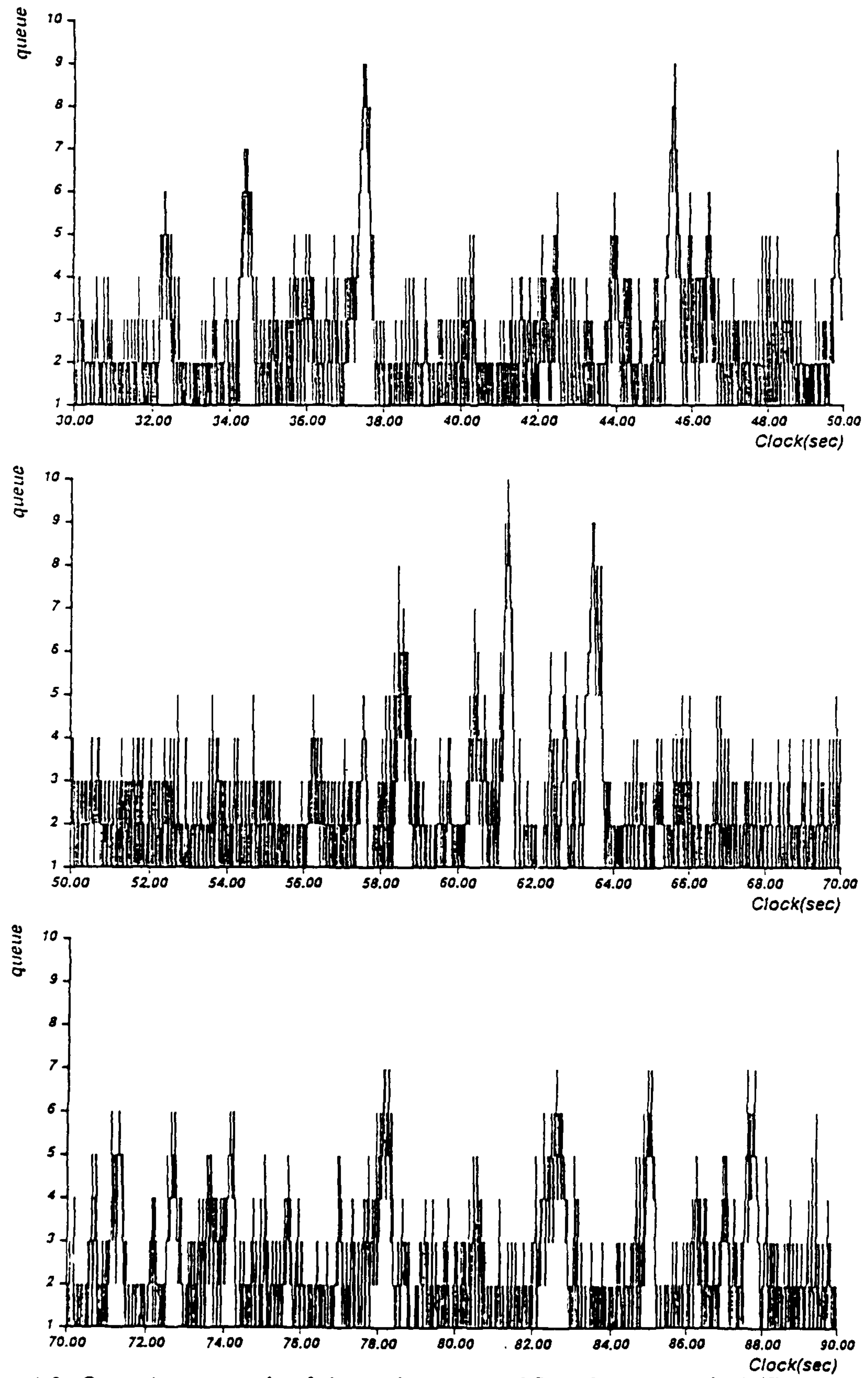


Figure 4-3: One minute sample of the packet queue of Switch-2, channel-12 (Flow Network).

and the average queueing delay of each group below. Here the queueing delay is the waiting time each packet experienced in the queue(s), excluding its own transmission time. The hop count of the flow is the number of the switch-to-switch channel(s) it crosses. Those channels are the bottleneck points in the network.

	Average Throughput (packets/sec)	Average Queueing Delay (msec)
1-hop flow	9.59	7.76
2-hop flow	9.58	14.58
3-hop flow	9.62	22.37

Notice that the actual average throughput is slightly (about $\sim 4\%$) lower than the requested value, due to the moving-average control that we discussed in Chapter 3; so is the actual channel utilization as compared to the control value $U_{capacity}$. Converting the packet waiting time to the queue length, because it takes 5 msec to transmit a 250-byte packets over a 400 Kbps channel, the average waiting time agrees with the average queue length shown above (the queue length counts the packet being transmitted as well).

Summarizing the test results, we see that:

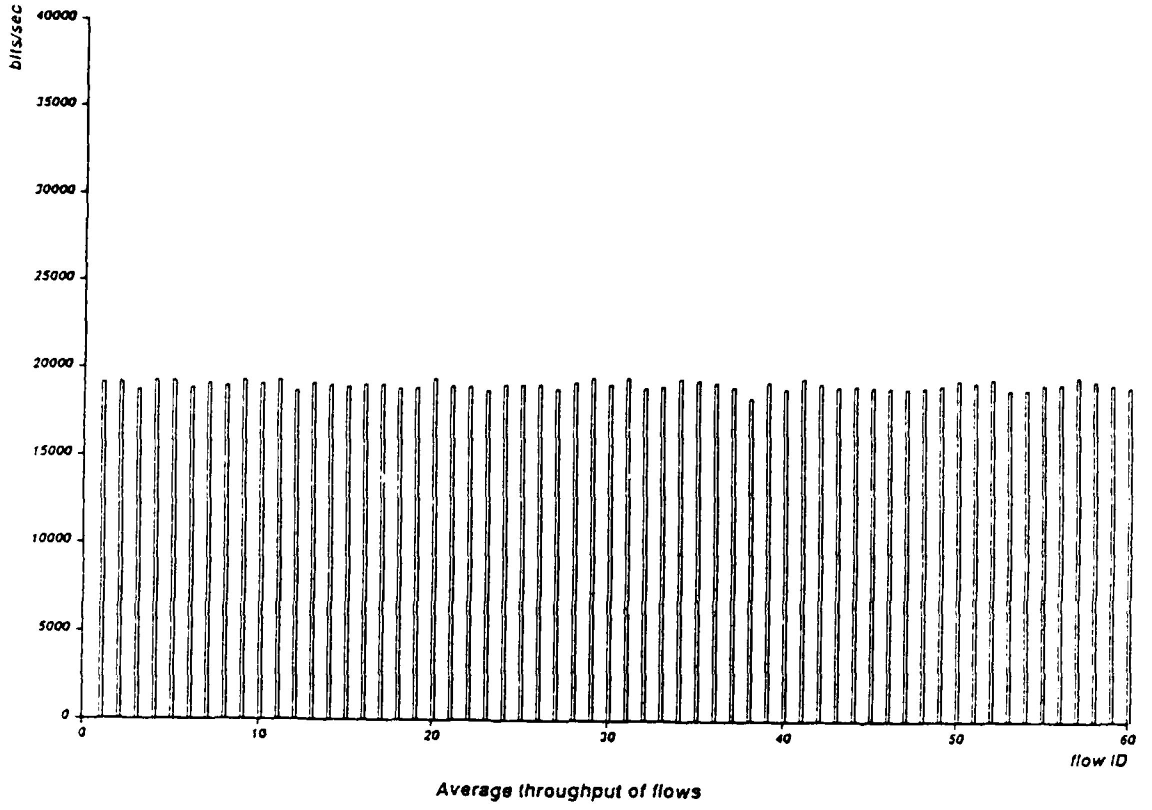
- The network meets the flows average throughput requirement.
- The average queueing delay is low (as a point of reference, an M/D/1 queue's average length under the same utilization would be around 4 packets, or the average waiting time 15 msec).
- The network load is stable (as shown by the queue graph) and congestion free.
- The network provides a fair service, independent of flows' path lengths.

Flows with Diverse Throughput Requirements

The test condition here is changed to flows with different throughput requirements, as given below.

Diverse Throughput Rate of Flows (packets/sec)

Throughput	Flow ID
50	1, 18, 35
30	8, 25, 36
20	3, 12, 20, 29, 37
10	2, 4, 5, 6, 7, 9, 10, 11, 13, 14, 15, 19, 21, 22, 23, 24, 26, 27, 28, 30, 31, 32
5	16, 17, 33, 34



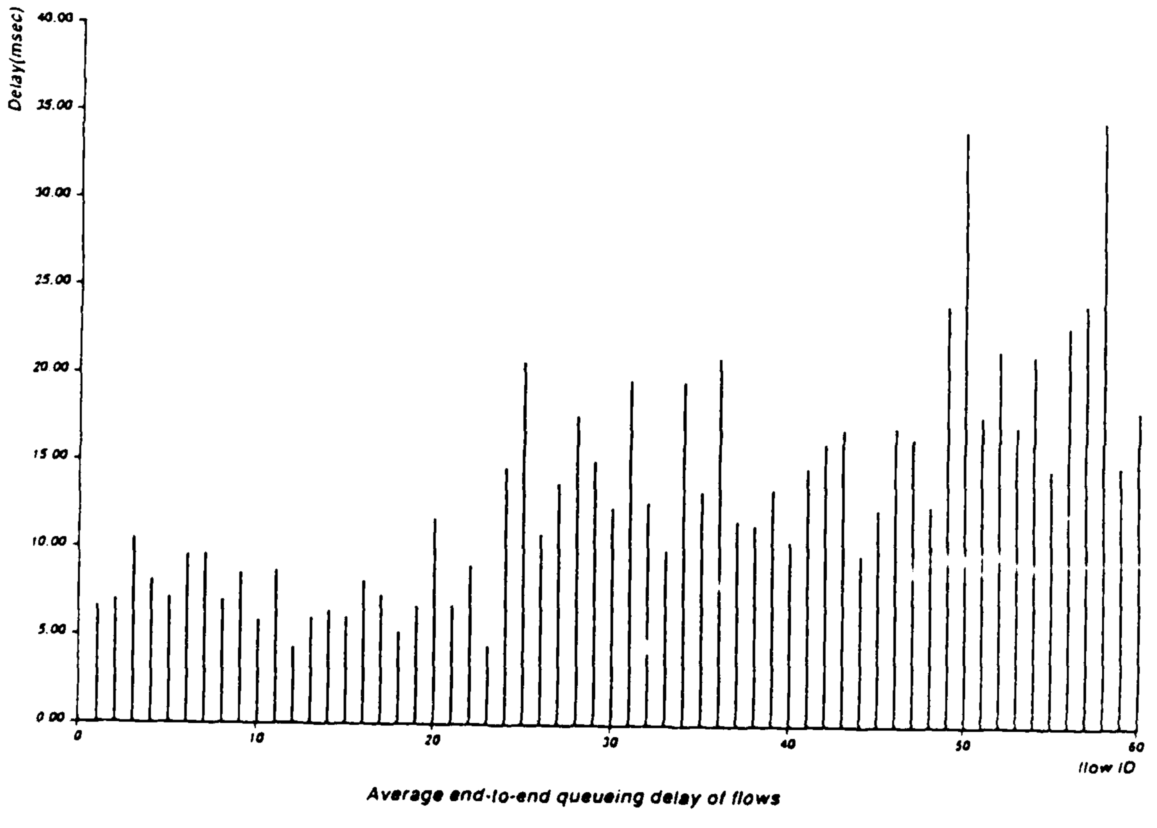


Figure 4-4: The average throughput and end-to-end queueing delay of flows.

Among the total of 37 flows, 1 ~ 17 are 1-hop flows, 18 ~ 34 are 2-hop flows, and the rest 3-hop. All the other conditions are the same as in the last test. The test simulated a 10 minute run of the real system and the results are presented in the same way as before.

Measurement statistics with diverse throughput flows

Switch	Total	Packet	Channel	Utilization		Queue Length		
ID	Forward	Loss	ID	mean	dev	mean	dev	99-tile
2	268138	0	12	0.81	0.14	2.29	1.41	8
3	268748	0	12	0.82	0.12	2.34	1.60	9

Effective throughput: 564 packets/sec

Total loss: 0

Again, we show the average throughput and queueing delay of the flows by the path length groups.

Flow performance with diverse throughput rate

	Average Throughput(pkts/sec)					Average Queueing Delay(msec)				
Desired rate	50	30	20	10	5	50	30	20	10	5
1-hop	48.2	29.0	19.3	9.6	4.7	5.6	4.2	5.2	10.8	12.3
2-hop	48.3	28.8	19.0	9.6	4.9	8.5	8.3	7.8	17.6	21.3
3-hop	47.8	29.0	19.4			9.5	8.0	10.7		

The above results show that the Flow Network satisfies the users with their expected throughput; different path lengths show no effect. The different throughput rates do have certain effect on the average queueing delay though: low throughput flows seem to experience a higher queueing delay. This is because their VirtualClocks tick by bigger steps, one packet arrival may advance the VirtualClock so much that the next packet has to wait to let one or more packets from high-speed flows, which arrived in burst, pass by first.

Flow Performance in the Presence of Misbehaving users

Here the test condition is changed back to the original 60 flows, except that every 6th flow is now a misbehaving user: it sends at a speed of 5 times the specified rate, and does not listen to network control commands. The test simulated a 5 minute run of the real system.

Measurement statistics in the presence of misbehaving users

Switch	Total	Packet	Channel	Utilization	Queue Length		
ID	Forward	Loss	ID	mean	mean	dev	99-tile
2	163807	9948	12	1.0	47.4	7.65	65
3	163881	9989	12	1.0			

Effective throughput: 680 packets/sec

Total loss: 47106 packets (all from misbehaving users)

Throughput and delay of flows in the presence of misbehaving users

	Average Throughput (packets/sec)	Average Queueing Delay (msec)
1-hop flow	9.59	8.33
2-hop flow	9.64	14.82
3-hop flow	9.65	16.36

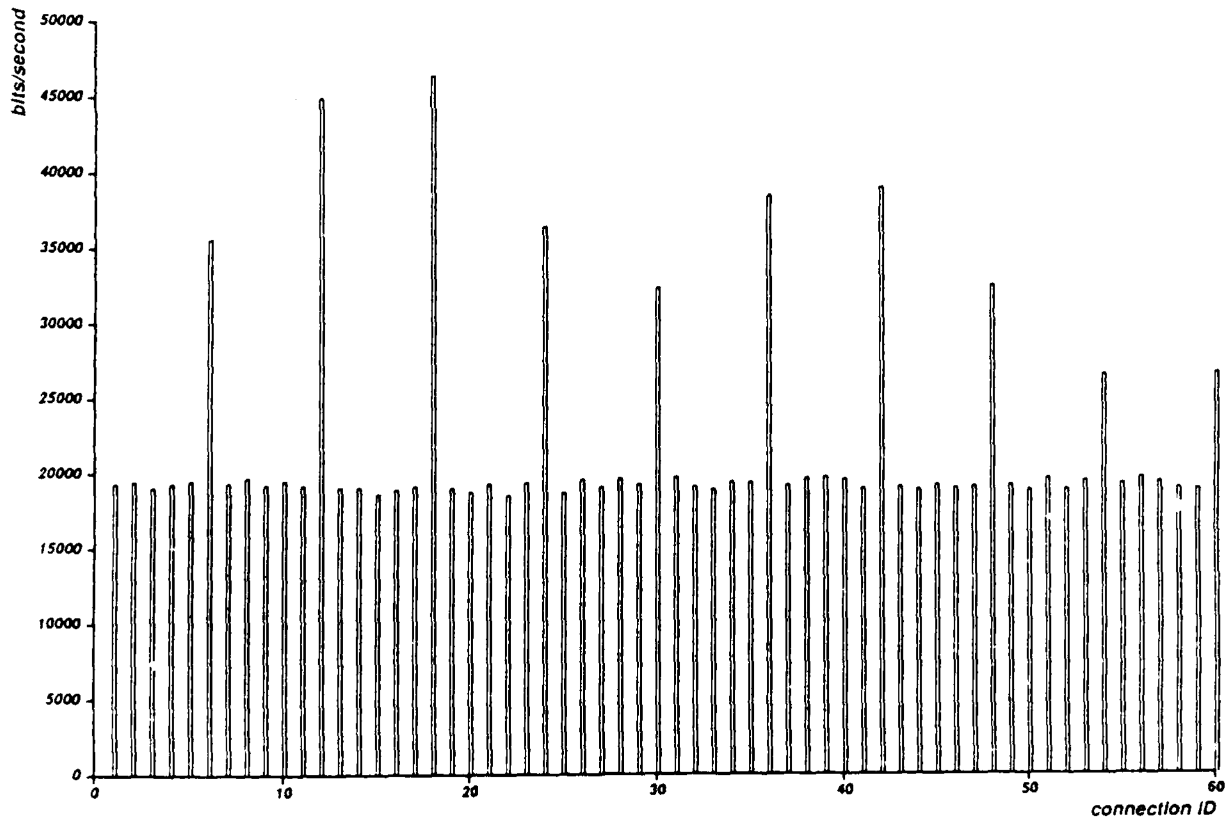


Figure 4-5: Flow throughput with misbehaving users.

Figure 4-5 gives a graphic show of the average throughput of all the flows.

Recall from Appendix-3A that when a switch runs out of buffer space, it drops the last packet from the longest queue; and that after a switch has sent a number of control commands to a misbehaving flow but observed no change of the flow's behavior, it will erase the flow from its flow table. In this test, because the misbehaving flows sent much too fast, their packets were put at the end of the queues; and because they did not listen to the control commands, they were quickly erased from the switches' flow tables. Further packets from these erased flow were treated as from unknown users and received the lowest priority in handling.

The test results show that normal flows are well protected from the few misbehaving users. They received the same throughput as in Test-FP-One, no one lost a single packet, even though the misbehaving users drove the channel utilization to 100%. Also notice that the queueing delay of the flows remains about the same as before; the 3-hop flows even receive a lower queueing delay. This is because the switches deleted the misbehaving flows, making regular flows see a lower utilization than in Test-FP-One.

4.4.2 TCP-SS Performance

Tests on TCP-SS were performed by repeating the same simulation runs as we did for the Flow Network, but with each flow replaced by a TCP connection. The same packet-train model and parameters were used in data generation unless otherwise specified. The receiver window size of each TCP connection was set to 10 packets except in the simulation run with diverse connection throughput, where the receiver window size was adjusted according to the desired throughput value.

TCP-SS with Homogeneous Connections

The goal of this test is to see how well TCP-SS can perform under the same traffic load as that in Test-FP-One. Data packets in the two tests were generated at exactly the same rate. The actual load here, however, was higher, because of the overhead of acknowledgment packets and the retransmissions needed to recover lost data packets.

The results of a 10-minute simulation run are presented in the same way as for the Flow Network. Below is the measured statistics; a one-minute channel queue sample is shown in Figure 4-6.

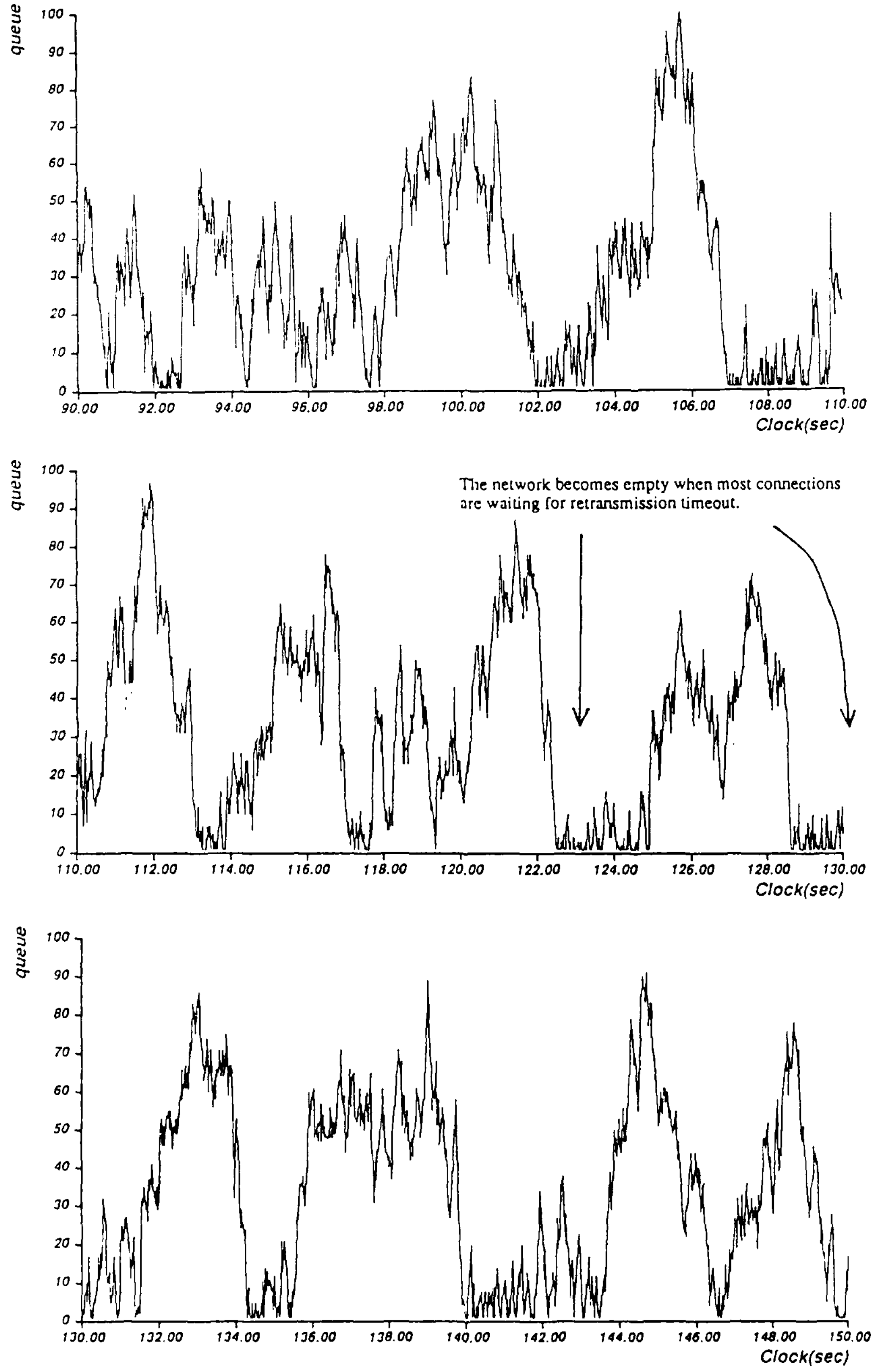


Figure 4-6: One minute sample of the packet queue of Switch-2, channel-12 (TCP/IP with Slow-Start)

Measurement statistics with homogeneous TCP connections

Switch	Total	Packet	Channel	Utilization		Queue Length		
ID	Forward	Loss	ID	mean	dev	mean	dev	99-tile
2	266945	5198	12	0.93	0.15	23.77	18.58	75
3	267397	4290	12	0.96	0.11	33.14	22.08	86

Effective throughput: 545 packets/sec

Total losses: 16135 packets(data packets 10728, the rest are acks)

Total retransmissions: 16373 packets

The performance of individual connections are summarized below. The average throughput, average queueing delay, packet losses, and retransmissions are shown in the three path length groups. A graph of the throughput, losses, and retransmissions of each connection is also attached (see Figure 4-7).

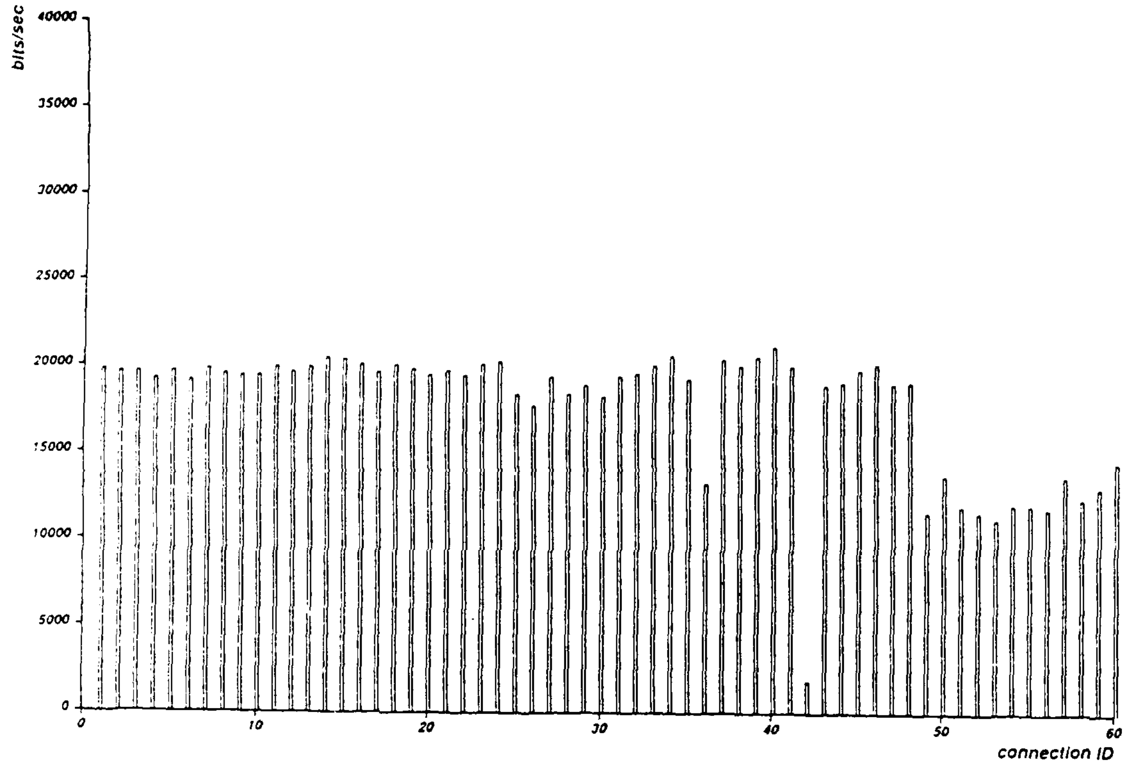
TCP Connection Performance

	Ave Throughput(pkt/sec)	Ave Queueing Delay(msec)	Loss(pkt)	Retrans(pkt)
1-hop	9.93	135.6	180	241
2-hop	9.24	219.6	178	272
3-hop	6.28	304.1	183	337

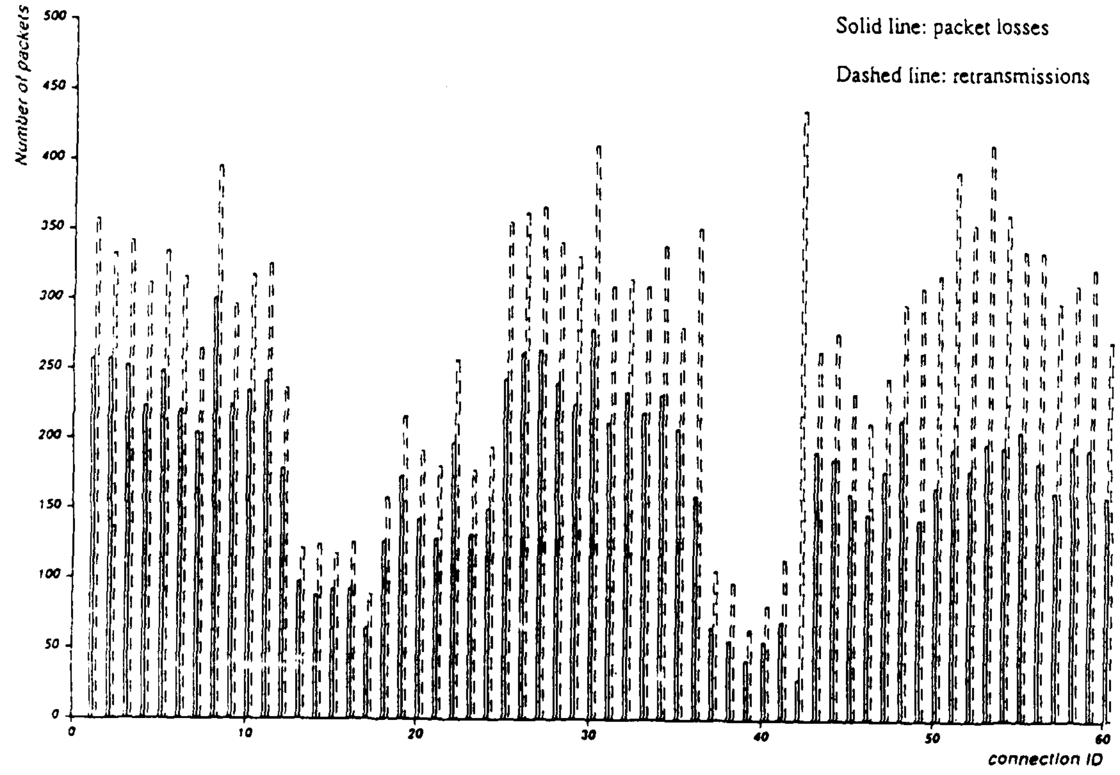
The measurement data and graphs suggest serious performance problems. We discuss them one by one.

Oscillating behavior of the network load. This phenomenon deserves special attention. When a switch is congested, packets from multiple connections may get lost. During the time that all the affected connections wait for a retransmission timeout, the network becomes empty. Because losses are synchronized by congestion, the timing of retransmission and the phase of the window close and reopen of individual connections are highly synchronized. As mentioned earlier, this is an example of control synchronization phenomena (end-point control in this case) discussed in Chapter 3. It results in high queueing delay variations and packet losses at peaks of the load resonance.

It is worth pointing out that the queueing behavior shown in Figure 4-6 is not the worst oscillation observed. Oscillation becomes worse when users have such a high transmission demand that packet transmission is limited only by the window flow control. We emulated such high demand with an infinite data source model where each connection always has data



Average throughput of TCP connections



Losses and retransmissions of TCP connections

Figure 4-7: The average throughput, packet losses and retransmission of TCP-SS connections.

ready to send, as is often the case in the ARPA Internet where the dominant load comes from large quantity data transfers. Figure 4-8 shows a packet queue sample with the infinite data source model.

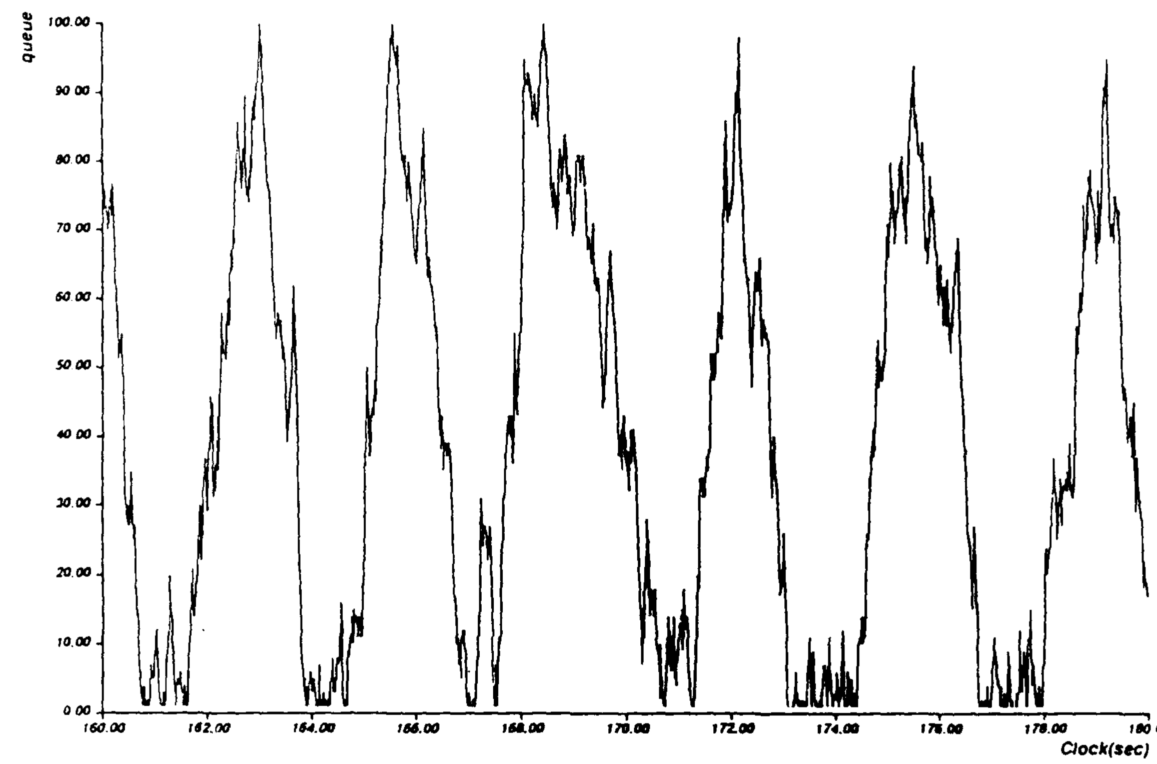


Figure 4-8: A 20-second sample of packet queue of Switch-2, channel-12 (TCP-SS with infinite data source)

Long queueing delays. This is the consequence of the packet queueing inside the network. Since window flow control tends to drive the network into full utilization, queueing delay is necessarily high. Slow-Start does not change this intrinsic feature of window flow control.

Packet losses. This is a consequence of the excessive utilization. Packets get accumulated in the network and create a high demand for buffer space; when the buffers overflow, packets are lost. As a comparison, Flow Protocol constrains data sources not to send faster than the bottleneck point can forward, and thus packets never accumulate inside the network, even at bottleneck points.

Unfair throughput due to path lengths. Connections 49 $\sim$ 60, which have a longer path than others, have a lower average throughput as well. This difference is because, after a period of congestion, connections with a shorter path can reopen the control window more quickly than those with a longer path. The throughput of 1-hop connections was not much higher than that of the 2-hop connections merely because their throughput was determined by the data generation rate — no connection generated more than 10 packets per second on average.

We argued in Chapter 2 that, because of the differences in connections' response time, it is difficult to achieve a fair service with an end-point control. Although the service unfairness in this particular test does not seem to be significant — due to the special test condition — it is indeed a serious problem with TCP-SS in general. The unfairness is evidenced by repeating the simulation test with a different data generator: each connection is given an infinite amount of data to send, and the transmission is only limited by the window mechanism. Below is the connection throughput measured from the test; a graphic presentation is given in Figure 4-9.

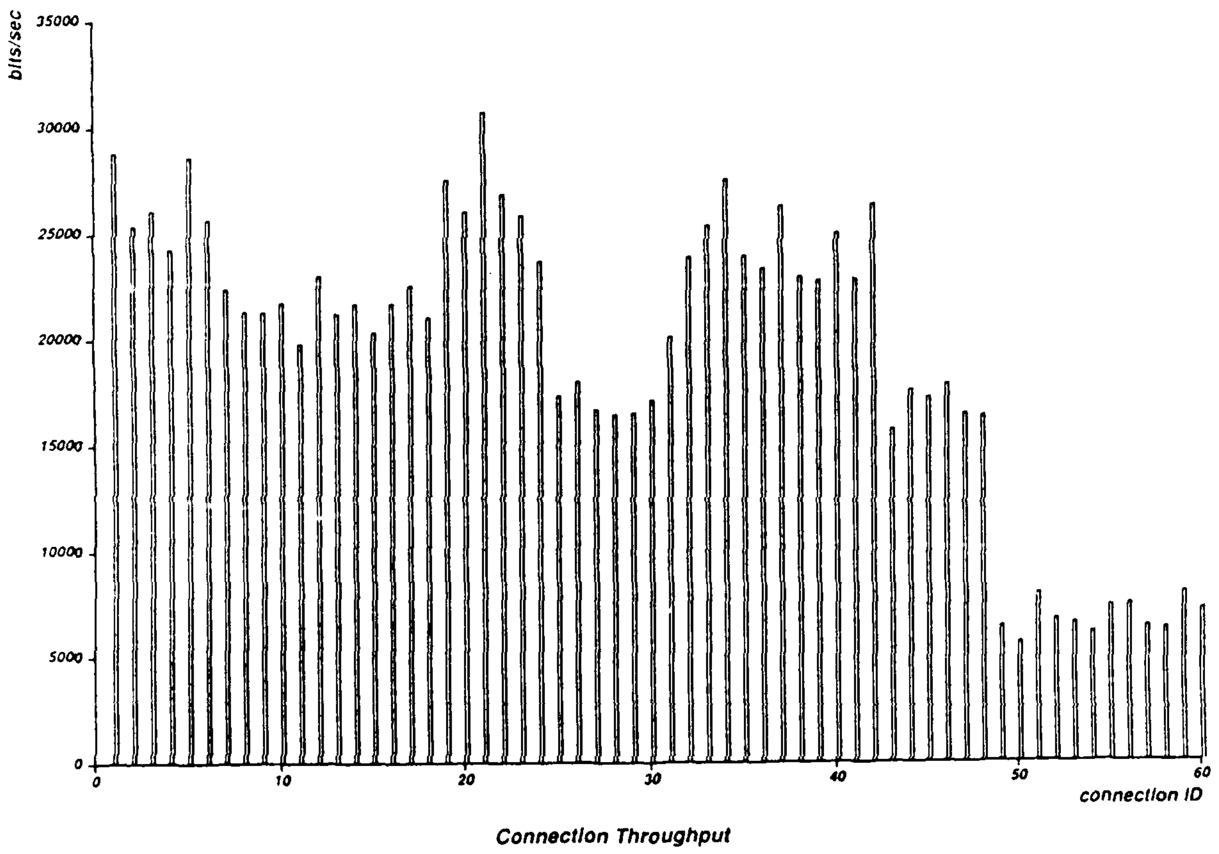


Figure 4-9: The average throughput of TCP connections with infinite data model.

TCP connection performance with infinite data source

	Ave Throughput(pkt/sec)	Loss(pkt)	Retrans.(pkts)	Retrans/loss
1-hop	12.0	494	624	1.26
2-hop	10.3	302	422	1.40
3-hop	3.4	215	403	1.87

We see that, although shorter-path connections have higher packet losses, their throughput is still higher because they recover losses more quickly; and their retransmission to loss ratio is lower because their round-trip-time has smaller variance and hence their retransmission timer makes fewer false alarms.

Compared to earlier TCP implementations, TCP-SS has significantly improved certain aspects of the performance measure. Superfluous retransmissions, which used to be a serious problem with earlier TCP implementations, are largely eliminated. Being a dynamic window adjustment algorithm, Slow-Start makes the selection of the original window size less critical to the connection's performance, as long as it is not too small. We also simulated the older BSD UNIX 4.3 implementation of TCP/IP which uses a fixed-size flow control window. With test parameters identical to those used above, Slow-Start reduced the total packet losses by 71%, and reduced the retransmission to data loss ratio from 2.9 to 1.5 (similar result is observed in SATNET measurement [11]). Overall, the effective throughput is increased by 26%.

End control, however, can improve performance only to a certain extent. The excessive queueing delays, the oscillating traffic behavior, and the uneven throughput due to path length differences that were observed in the UNIX 4.3 simulation showed little improvement in the TCP-SS simulation.

Once these problems are recognized, it is possible to make improvements with more intelligent end control algorithms. For example, one may insert a minimal time gap between receiving an acknowledgment and sending the next packet, to avoid driving the network to too high a utilization; or one may try to inject certain randomness into the retransmission timer to somehow alleviate the oscillating behavior of traffic.

We speculate, however, that such adjustments may not fit well to a wide range of changes in condition (such as changes in network bandwidths, RTTs, number of users, or even just different versions of protocol implementations), and the cost might be higher (e.g. requiring constant feedback signaling by ack returns or other methods) to achieve the same performance — if that is ever achievable — as compared to that of network control with a rate mechanism.

TCP-SS with Diverse Throughput Requirement

The results presented below are from a simulation run which was performed under exactly the same conditions as that in testing the Flow Network with diverse throughput flows, except that TCP connections were simulated in the place of flows. Data packets were generated at the same rate in the two tests. The receiver window size of each TCP connection was set to proportional to its average packet rate, i.e. the window size was 10 if the connection generated 10 packets/sec, 20 if the connection generated 20 packets/sec, and so on.

Measurement statistics with diverse TCP connections

Switch	Total	Packet	Channel	Utilization		Queue Length		
ID	Forward	loss	ID	mean	dev	mean	dev	99-tile
2	258924	962	12	0.87	0.22	22.56	21.12	83
3	259846	1252	12	0.93	0.15	27.87	20.92	82

Effective throughput: 549 packets/sec

Total loss: 3797 packets

Average throughput and queueing delay of diverse TCP connections

	Throughput (packets/sec)					Queueing Delay (msec)				
Desired rate	50	30	20	10	5	50	30	20	10	5
1-hop	49.8	30.1	19.9	10.1	5.0	61	56	105	89	122
2-hop	48.7	29.2	19.9	10.2	4.9	134	135	194	159	208
3-hop	28.5	20.1	18.4			192	309	311		

The above data show that the 1-hop and 2-hop connections received the desired throughput. The throughput of high data rate, 3-hop connections, however, looks unacceptable: the connection desiring 50 packets/sec received only a bit above half of that rate, and the connection desiring 30 packets/second received just 2/3 of it. In this test, although packets were generated by the packet train model at specific average rates, connections often had packets accumulated at the source when the control window size was small; they sent the packets in bursts when the window opened up. Packet bursts tend to cause temporary congestion and packet losses. Shorter path connections recovered the losses and reopened the window more quickly, getting more packets through before the next congestion point.

TCP-SS With Misbehaving users

The following results are from a simulation run under the same condition as that in testing homogeneous TCP connections, except that every 6th connection is a misbehaving user. Misbehaving users have an infinite amount of data to send and use a Go-Back-N retransmission strategy. The data generator of good connections was the packet train model with an average rate of 10 packets per second. The simulation ran for 10 minutes.

Measurement statistics in presence of misbehaving users

Switch	Total	Packet	Channel	Utilization	Queue Length		
ID	Forward	loss	ID	mean	mean	dev	99-tile
2	276450	12889	12	0.93	22.57	17.18	69
3	276224	13010	12	0.95	24.91	17.13	70

Effective throughput: 549 packets/sec

Total loss: 32827 packets

Average throughput of TCP connections in the presence of misbehaving users

	Normal Connections			Misbehaving Users		
	Throughput(p/s)	Loss(pkt)	Retrans(pkt)	Throughput	Loss	Retrans
1-hop	8.85	408	532	20.56	674	1679
2-hop	8.49	231	366	15.50	365	1160
3-hop	3.65	197	375	7.28	419	1344

Figure 4-10 shows that the 10 misbehaving users received about twice the throughput of the regular connections on average. The good connections did not get the desired throughput. In particular, the 3-hop connections only received one-third of the desired throughput, and couple of good users (Connections 19 and 47) were almost completely shut out. We would also like to remind the reader that the so called "misbehaving" users here are merely an earlier implementation of the same protocol.

4.4.3 Comparison between the Flow Network and TCP-SS

Here we discuss the most noticeable performance differences — queueing delay, throughput, service fairness, load oscillation, and service vulnerability — between the Flow Network and TCP-SS that we observed from the above simulation results.

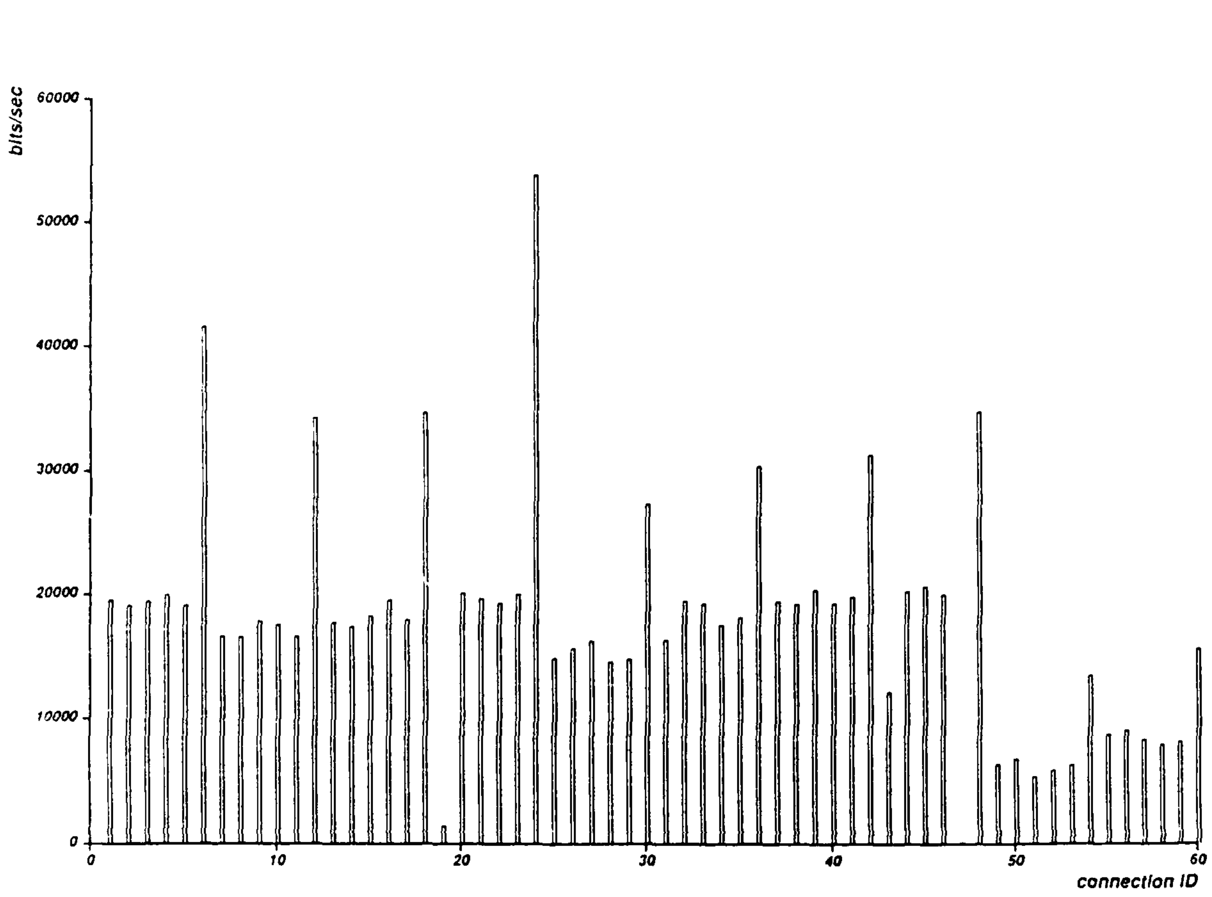


Figure 4-10: TCP connection throughput in the presence of misbehaving users.

Queueing Delay. The average end-to-end queueing delay in TCP-SS tests is about 10 times the delay experienced by the flows. The Flow Network maintains a proper utilization to keep queueing delay within a bound.

Throughput. The total throughput of the Flcw Network is not much higher than that of TCP-SS. We consider, however, that in this case the TCP-SS throughput is no longer a valid measure of performance, because it is associated with such undesirable long delays.

Fairness. The Flow Network meets flows' desired throughput independent from their path lengths, but TCP-SS does not. Under heavy load, the throughput of each TCP connection is sensitive to its path length — the longer the path, the lower the throughput.

Traffic Oscillation. The oscillation in TCP-SS is due to synchronized reactions to congestion. Control synchronization is a common danger in distributed control systems. Chapter 3 proposes randomization as a solution.

Because the only input to the Slow-Start algorithm is observations from the network, however, it is difficult for the connections to tell which are commonly observed events (so that randomization in reaction is needed) and which are specific signals to individual connections (so that reactions should be prompt). For instance, when its retransmission timer goes off, a TCP connection cannot tell whether it is due to congestion, which belongs to the first category, or due to random transmission errors, which belongs to the second. Therefore, it seems particularly difficult to overcome synchronized control actions in this case.[7]

Service Vulnerability. The service vulnerability in TCP-SS is due to the network's lack of discrimination against misbehaving users, which in turn is a result of the network being a stateless system.

Simulation of TCP-SS also exposed its other drawbacks. First, as a by-product of the TCP window mechanism, acknowledgment packets double the total number of packets traversing the network, thus doubling the demand on the switch processing speed.

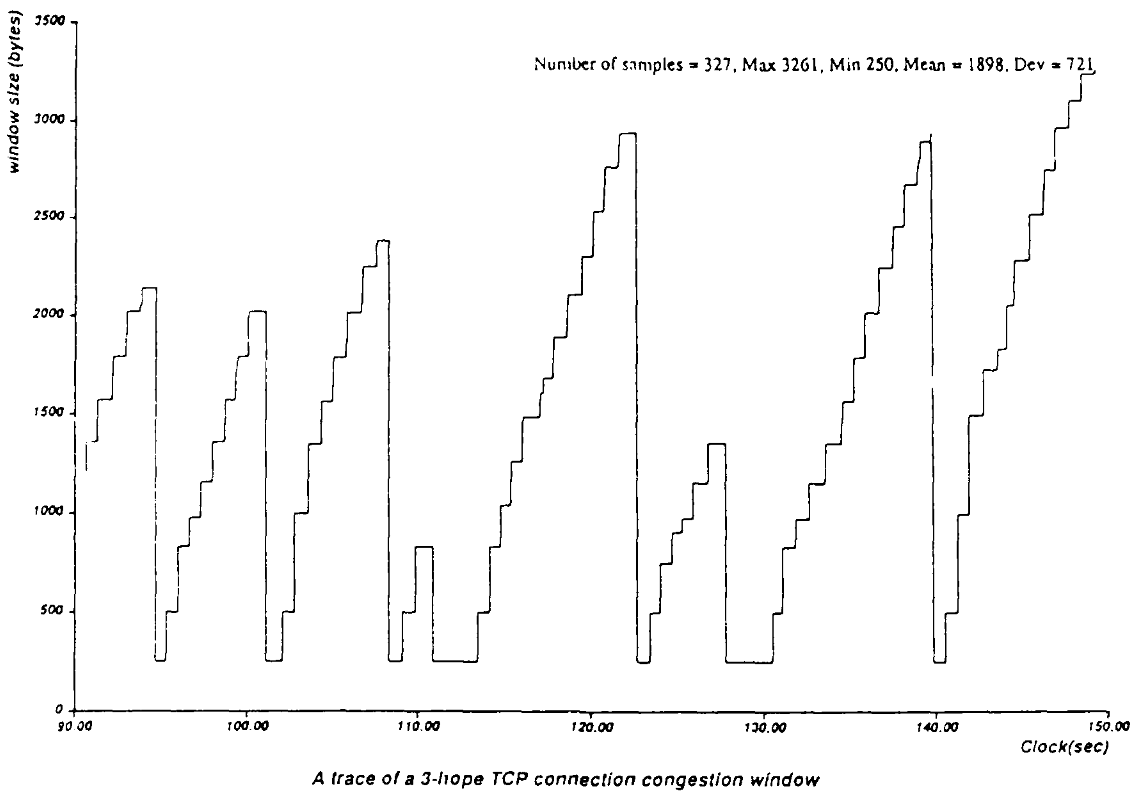
Secondly, from the trace of the control window size of a 3-hop TCP connection and the corresponding transmission behavior shown in Figure 4-11, we see that the connection's control window keeps repeating the cycle of creeping up until it encounters congestion and then closing down. As a result, the connection's sending rate is also constantly changing with the control window size, enforcing a transmission pattern that would be unacceptable to many real-time applications.

4.4.4 Random-Drop as a Stateless Congestion Control Algorithm

Four simulation tests were performed to examine the effectiveness of Random-Drop in a TCP/IP environment. We wanted to see if Random-Drop can prevent misbehaving users from receiving a higher effective throughput than other connections.

In the four simulation tests, all TCP connections are assumed to have an infinite amount of data to send; and when a switch decides to drop a packet, it picks a random one from a packet queue that is the longest one among all the packet queues in front of its output channels. Each test simulated a 10 minute run of the real system.

[7]Look at the queue oscillation example again: it seems impossible to eliminate synchronized packet losses due to congestion. As a result, connections' wait for retransmission timeout will always be synchronized. So the solution is to eliminate congestion. Congestion, in turn, is the result of aggressive probing, the information acquisition tool used by Slow-Start — a connection does not know whether the maximum affordable throughput has been reached until it observes packet losses.



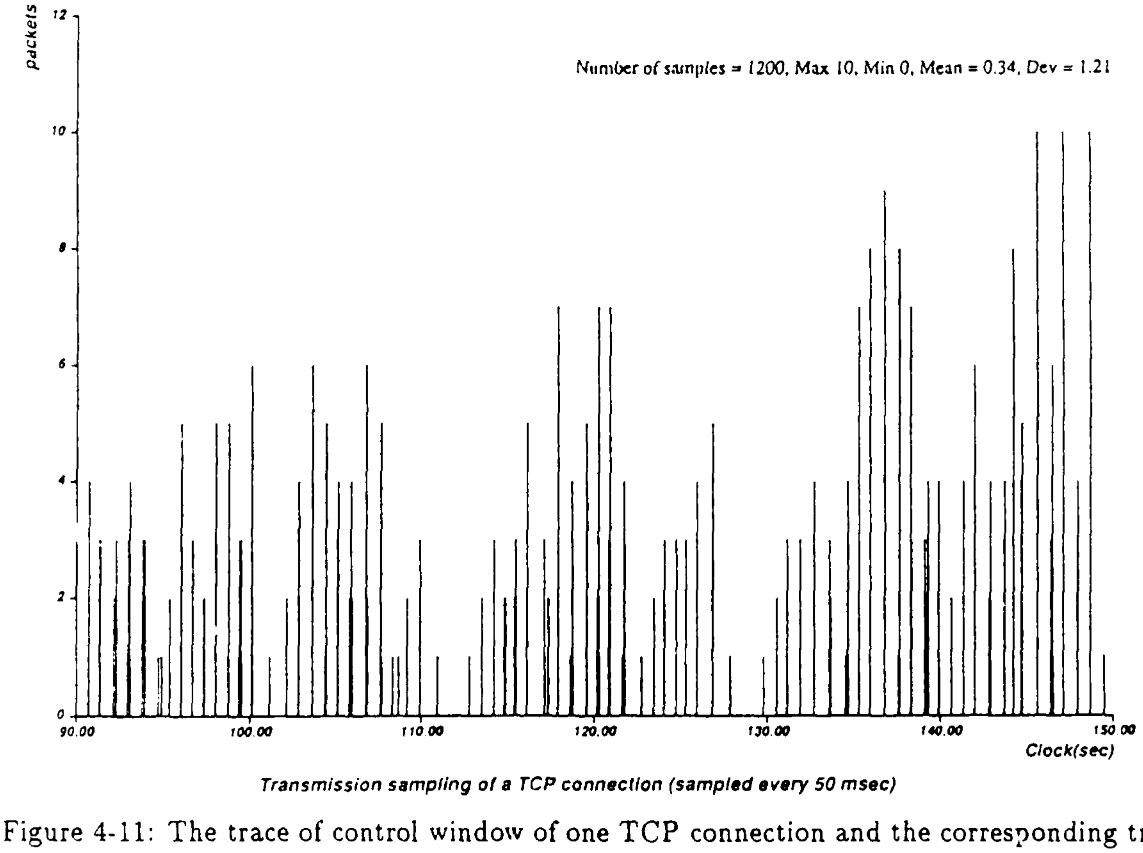


Figure 4-11: The trace of control window of one TCP connection and the corresponding transmission behavior.

In the first test, all the connections perform normally, and a switch randomly drops a waiting packet when it has no more buffer space for a newly arrived one. The second test is the same as the first one, except that when the switch buffer becomes 50% full, the switch may randomly drop a waiting packet with a probability of 0.02 upon a new arrival. The third and fourth tests are repetitions of the first and the second, respectively, except that every 6th connection is a misbehaving user.

It was speculated that, by starting dropping packets earlier with a small probability, only a sub-group of the connections — most likely, those sending faster — would be forced to stop to wait for a retransmission timeout. Therefore, misbehaving users would probably lose more packets and be stopped more often, taking less advantage of others.

The effective throughput of all the connections of the four tests is shown in Figures 4-12 and 4-13. These results illustrate four points:

1. The results from the first two tests suggest that, even when all connections behave correctly, Random-Drop provides no noticeable improvement on uneven throughput caused by uneven path lengths.

2. The last two tests show that Random-Drop failed to prevent misbehaving users from taking an unfair share of network resources. Despite the fact that the misbehaving connections received higher losses than the connections performing Slow-Start (the losses are not shown), the *Go-Back-N* retransmission strategy allows them to recover losses quickly while Slow-Start connections are backing up their control window size.

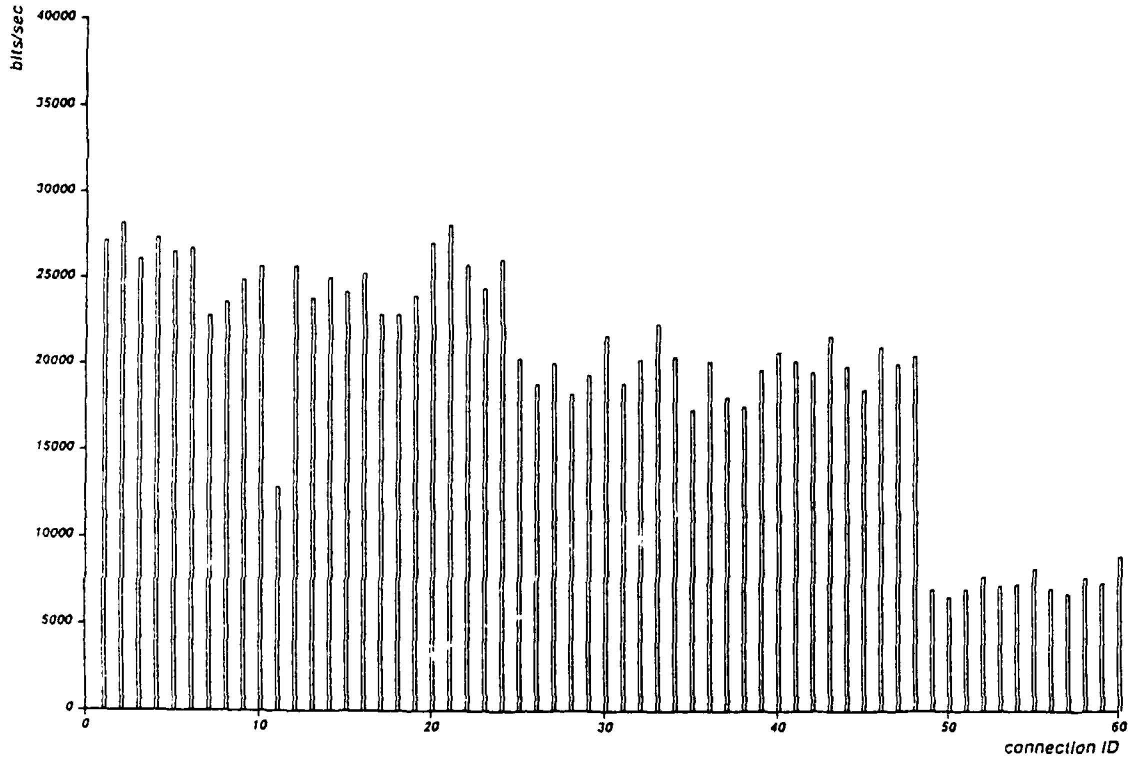
3. The second and fourth tests failed to show any noticeable improvement by starting Random-Drop early, as had been speculated.

4. Comparing the throughput of Connections 6, 12, 18, 24, ..., and 60 in the first test with that in the third and fourth tests, we see that a connection does get a higher throughput by not obeying the Slow-Start flow control rules.

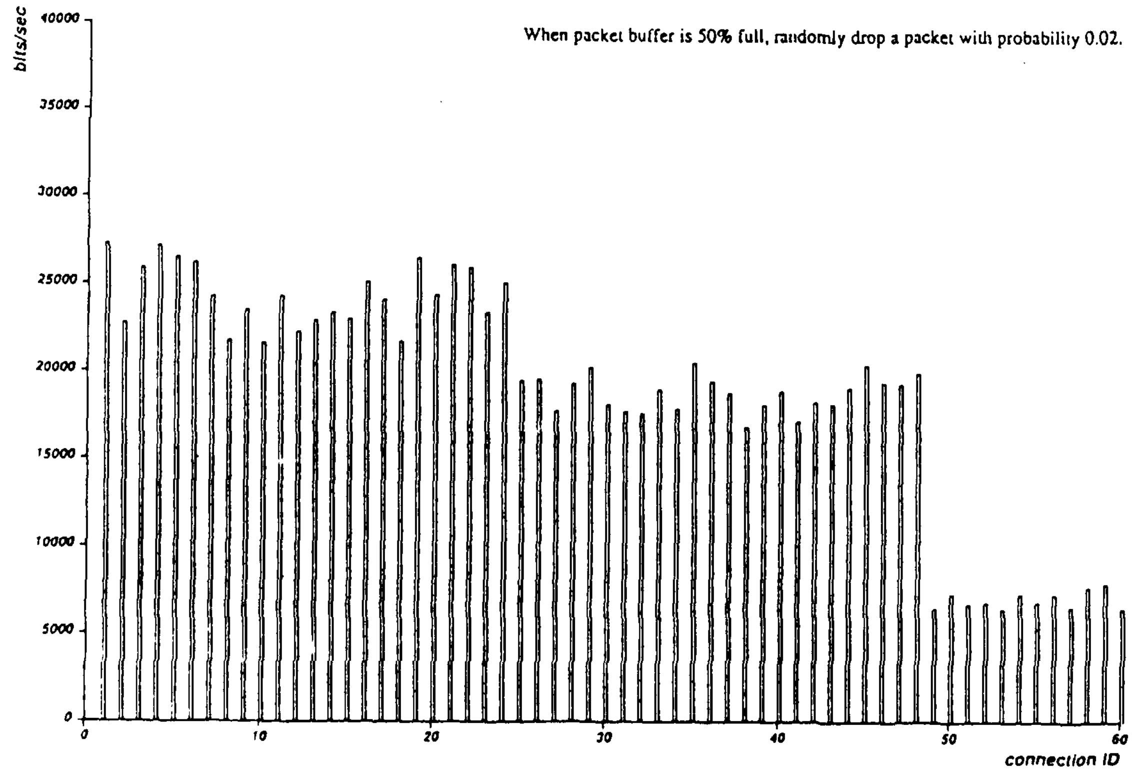
It was also observed during the simulation run that Random-Drop did not improve the oscillating behavior of the load.

Close observation of the simulation run explains why starting Random-Drop early did not bring performance improvement. It turned out that, much like an object with a large mass in motion, the "inertia" of the network traffic is so big that even long after the switch enters the dropping region, packets keep coming at the same speed as if there were no dropping, resulting in most connections, including well-behaved ones, getting packet lost.

The traffic "inertia" can be explained by the operation of window flow control. Suppose that the Nth packet of connection C is lost. C does not stop until the $(N + W - 1)$th packet is sent, where W is the window size (assuming the other direction of the path is not blocked

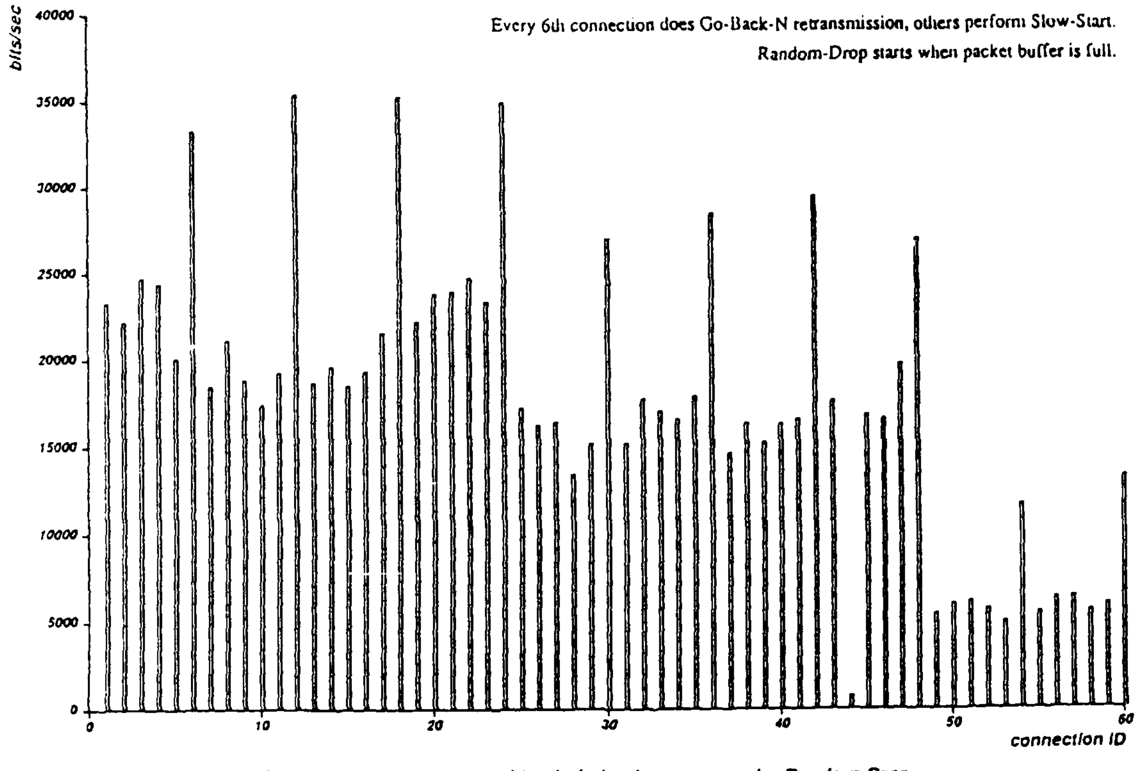


TCP connection throughput under Random-Drop

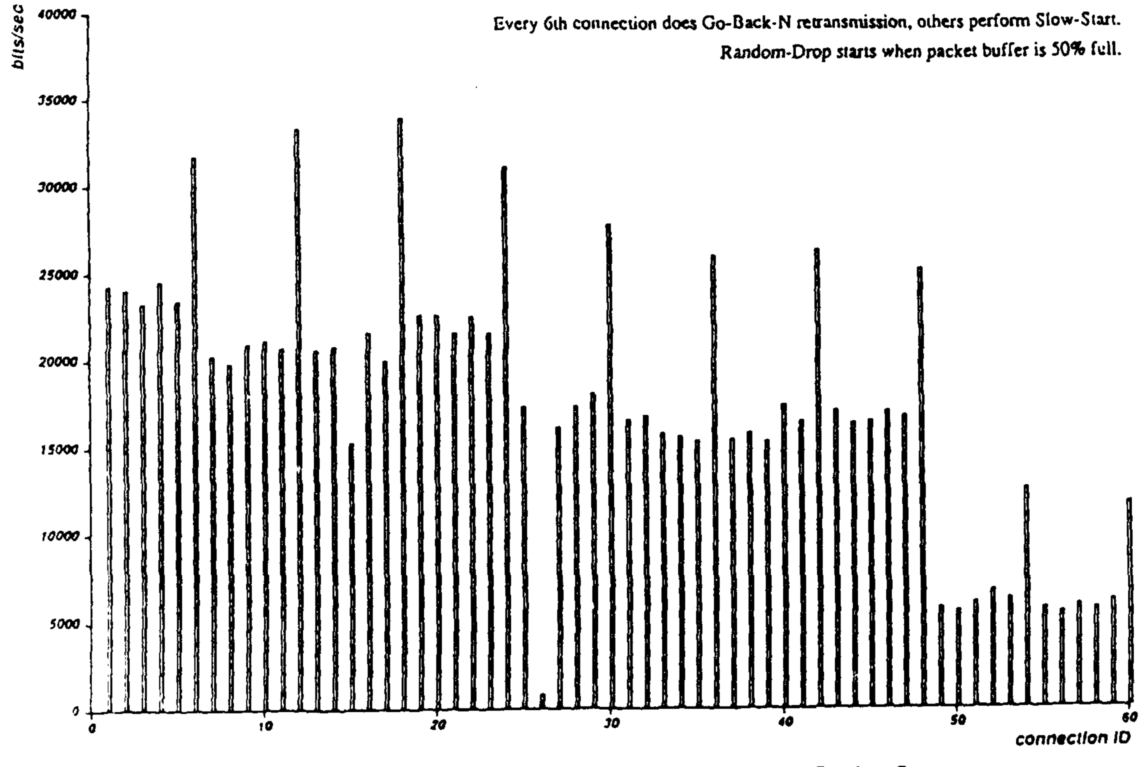


TCP connection throughput under early Random-Drop

Figure 4-12: TCP-SS connection throughput with random drop (all good users).



TCP connection throughput with mis-behaving users under Random-Drop



TCP connection throughput with mis-behaving users under early Random-Drop

Figure 4-13: TCP-SS connection throughput with random drop (with misbehaving users).

so that all the acknowledgments returned successfully). The "inertia" of one connection is its window size, and the inertia of the network load can be computed from the number of active connections and the window size of each. According to this analysis, when a switch detects overload and starts random dropping, the probability of making a drop has to be extremely small to avoid hurting the majority of connections during a rather long period when the traffic is still in the original "motion". The buffer space also has to be large enough to avoid overflow during this period.

Random-Drop would be ineffective even if end users perform a rate-based flow control instead of the end-to-end window flow control used in the above tests. If we assume a simple case of every user sending at a constant rate, R_i, every packet has the same probability, P, of being dropped, and the throughput of a user would be $(1 - P)R_i$. That is, whoever sending at a higher rate would get more packets through.

4.4.5 Summary

One cannot conclude, by simulation results alone, that long and varying queueing delays, service unfairness, and system instability are intrinsic to end-point control mechanisms (because one can never exhaust all possible adjustments to prove none of them works). The above simulation results and discussions, however, suggest that this may indeed be the case.

4.5 More Tests on the Flow Network

This section presents the results from more advanced simulation tests conducted on the Flow Network. Our focus will be on latency control, the performance of short transfers, the stability of network control with dynamic load, and the effect of heterogeneous network channels.

4.5.1 Latency Control

Below we discuss the roles and effects of various control mechanisms, moving-average, VirtualClock, priority, and channel utilization on network latency. We will also discuss the issue of phase-locking.

Effect of the User Behavior Envelope

The original motivation for enforcing a user behavior envelope (implemented as a moving-average control at the data source) is to eliminate excessive burstiness. As a positive side-effect, we observed in simulation that the moving-average control also cuts off peaks (i.e. periods with intensive new data) in data generation, thereby helping depress occasional queueing delay peaks.

As a comparison, another simulation run was performed with the moving-average control turned off (Test-NoMA). The test condition is the same as that in Test-FP-One but with the last two flows removed in order to keep the channel utilization at the same level with Test-FP-One. Attached is a graph of a 1-minute sample of a channel packet queue (Figure 4-14); the reader may compare this graph with Figure 4-3, the one from Test-FP-One with the moving-average control. The measurement statistics of Test-NoMA are given below, together with that of Test-FP-One for a comparison.

Channel queue measurement with/without M.A. control

		with M.A.					without M.A.				
Switch	Channel	Utilization		Queue Length			Utilization		Queue Length		
ID	ID	mean	dev	mean	dev	99-tile	mean	dev	mean	dev	99-tile
2	12	0.86	0.11	2.64	1.8	10	0.85	0.15	3.73	3.90	19
3	12	0.86	0.11	2.61	1.7	9	0.86	0.15	4.37	5.03	25

From the difference in the channel queue distribution, it should be expected that the end-to-end delay in the two simulation tests would be different as well. Figures 4-15 and 4-16 show two samples of the end-to-end queueing delay of two flow pairs: the first one in each pair is from Test-NoMA; the second one from Test-FP-One.

The 99th percentile of the queueing delay of the 3-hop flows is given below (Test-FP-One has 12 3-hop flows, Test-NoMA has 10 of them), where T is the transmission time of one packet. The slight difference in utilization is due to the difficulty of accurately adjusting the average of statistical load. The 99th percentile of a 3-stage Erlang distribution is given as a reference point.

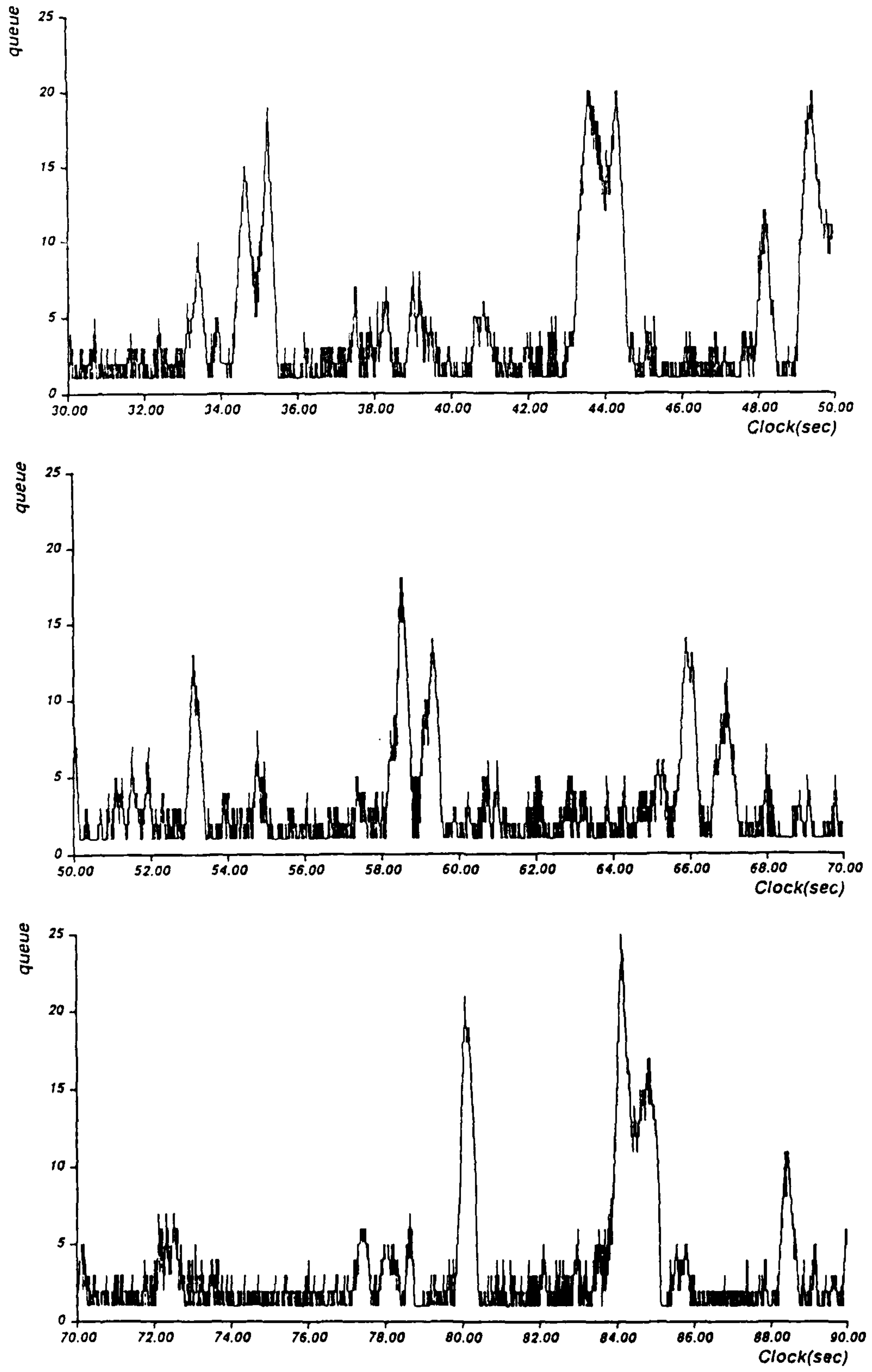
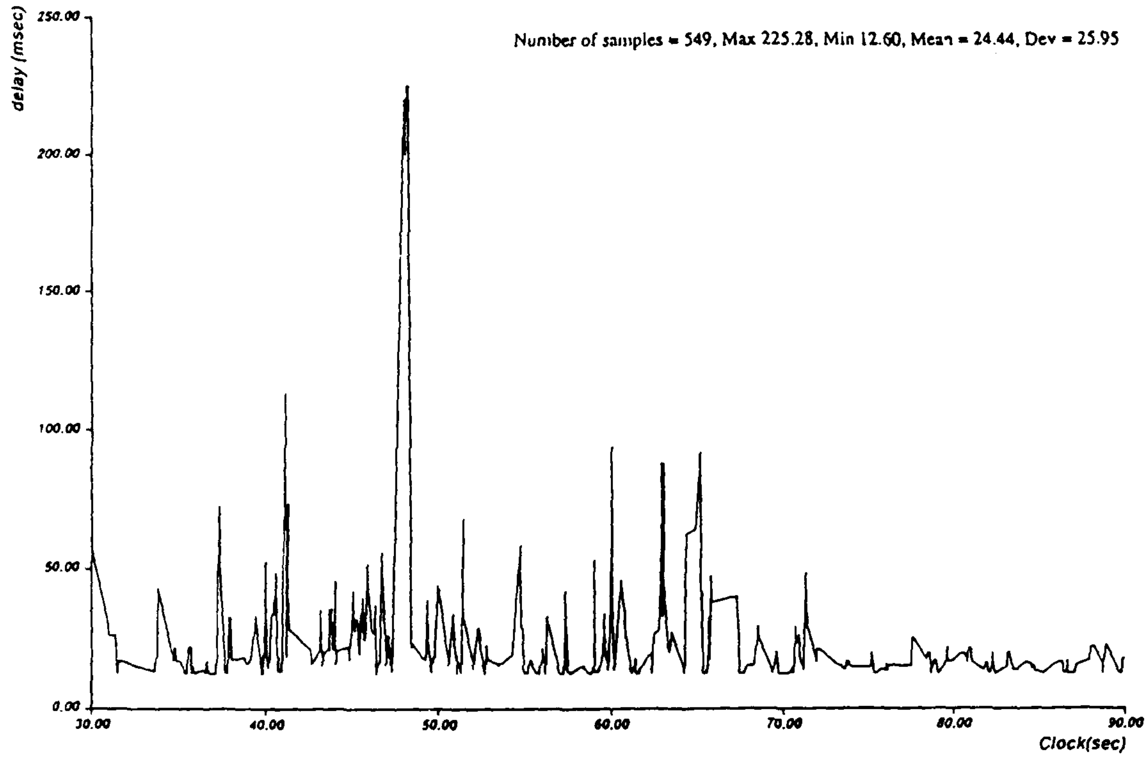
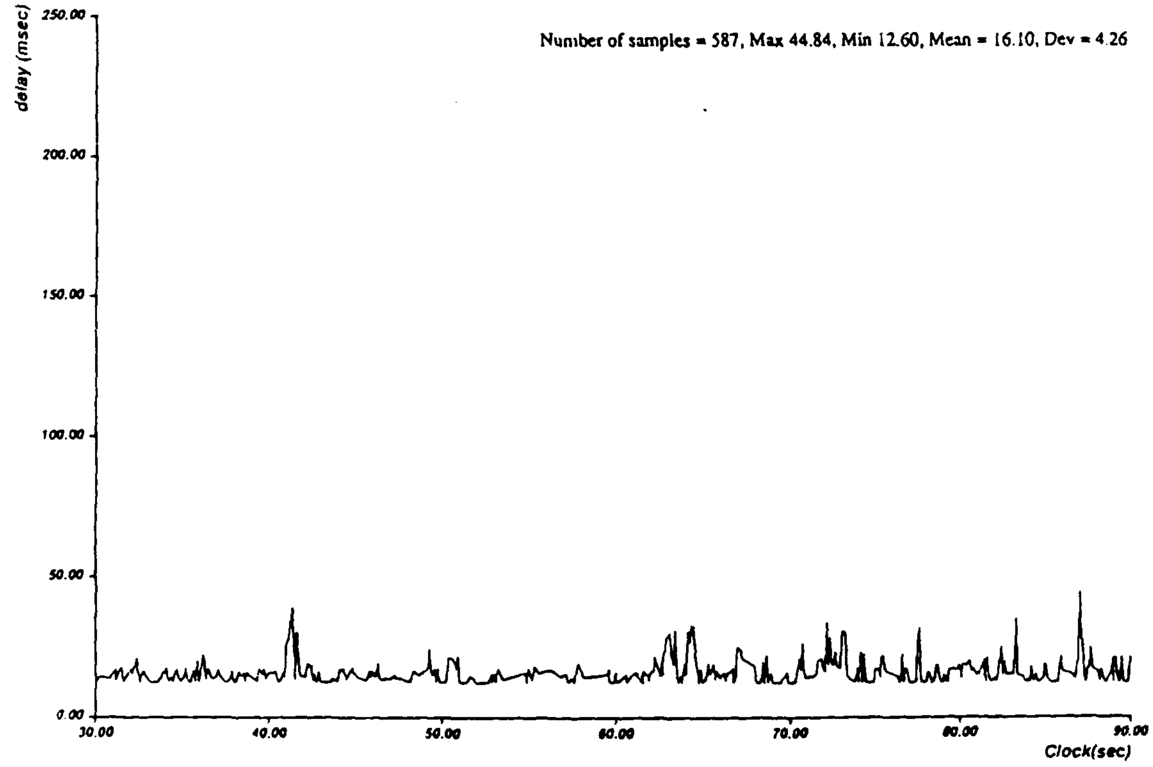


Figure 4-14: A 1-minute sample of packet queues without the moving-average control at flow sources.

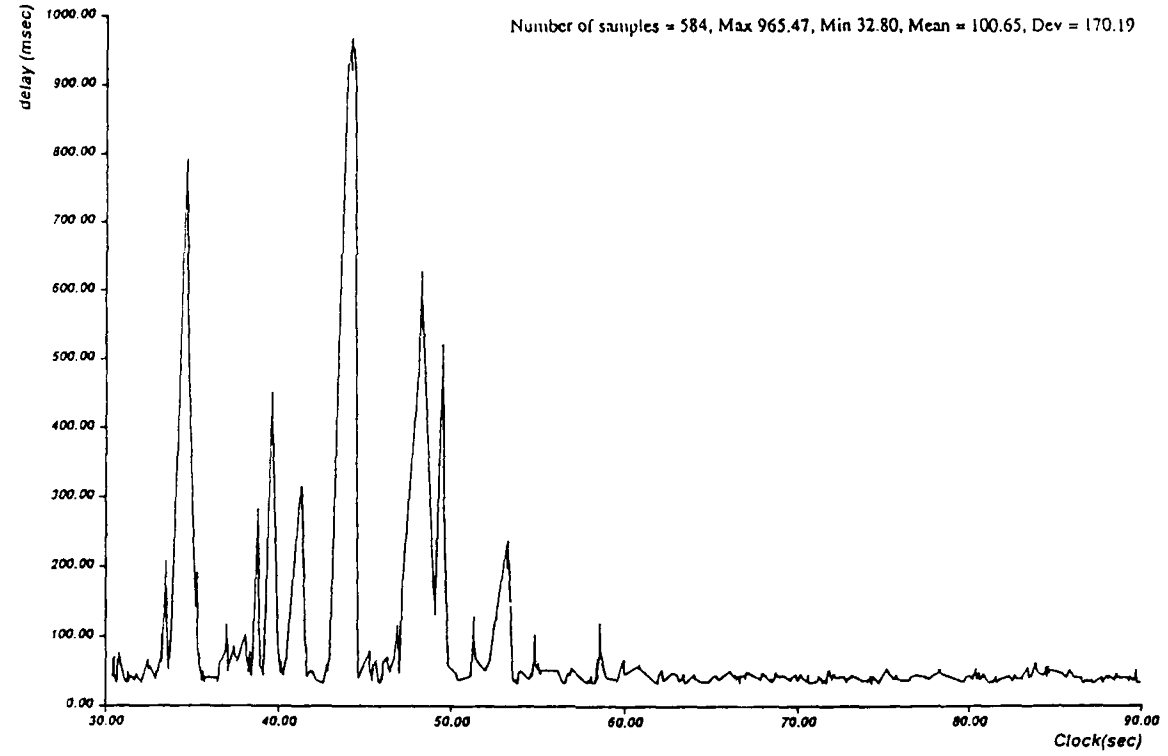


Flow-1 end-to-end delay (1-hop) without moving-average control

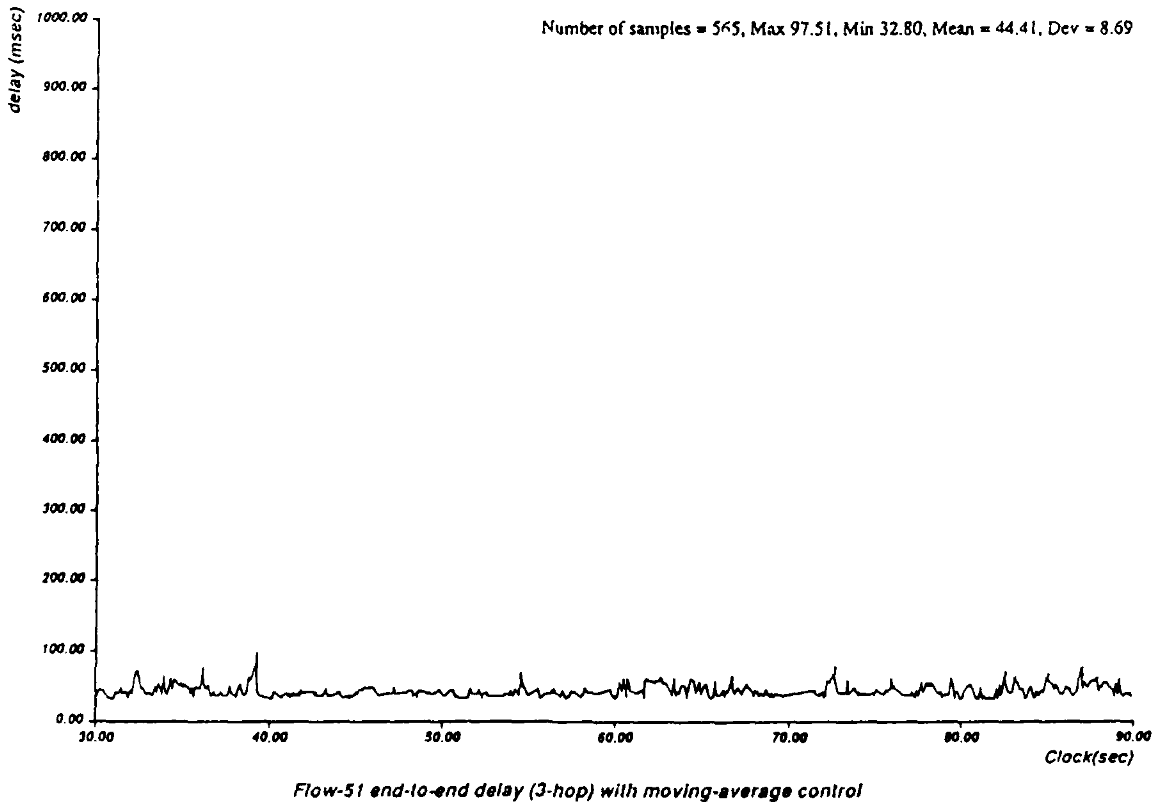


Flow-1 end-to-end delay (1-hop) with moving-average control

Figure 4-15: End-to-end packet queueing delay of 1-hop flows.



Flow-51 end-to-end delay (3-hop) without moving-average control



Flow-51 end-to-end delay (3-hop) with moving-average control

Figure 4-16: End-to-end packet queueing delay of 3-hop flows.

99th percentile of queueing delays of 3-hop flows

	Utilization	99-tile	99-tile Dev.
Test-FP-One (with M.A.)	0.86	42T	22T
Test-NoMA (without M.A.)	0.85	160T	89T
Erlang estimate	0.86	54T	

The variation in the measurement data is still too high to draw any quantitative relations between the average interval and the queueing delay distribution. Nevertheless, that all repeated simulation runs so far have shown consistent results suggests that the moving-average control indeed reduces the queueing delay variation substantially.

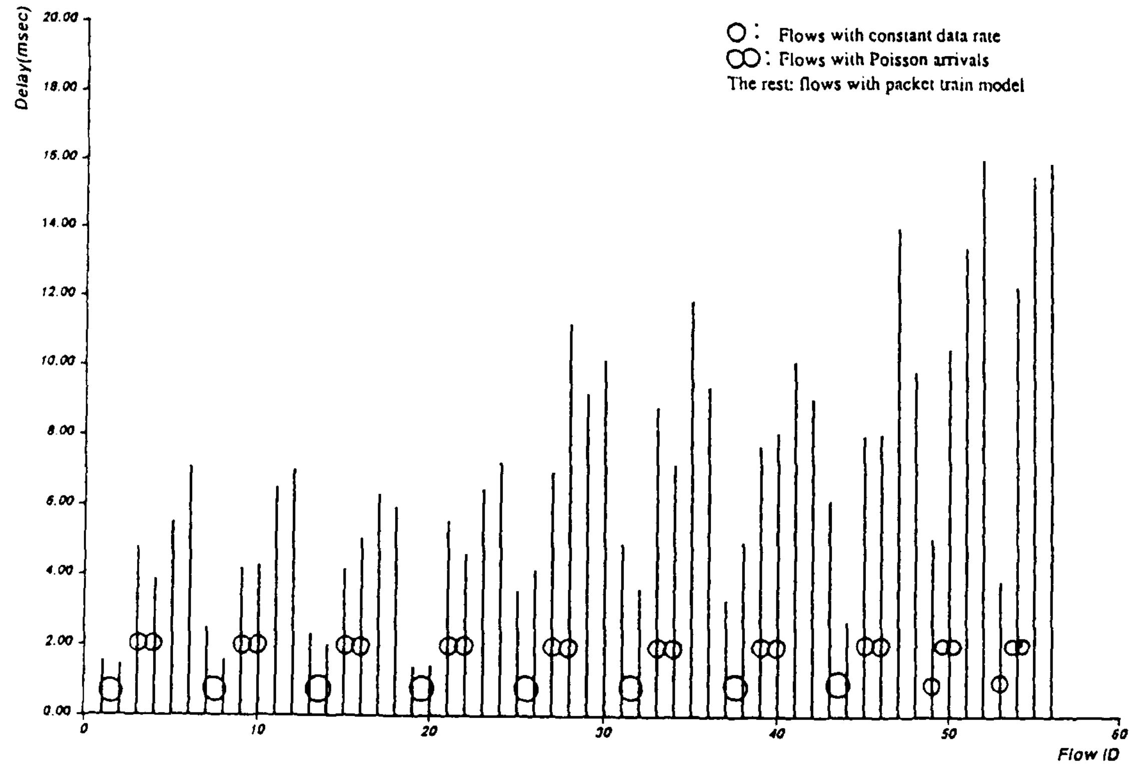
Effect of VirtualClock

The major role of VirtualClock is to meter the average volume of a statistical flow and to build firewall among flows in statistical multiplexing. It should be made clear that VirtualClock does not contribute directly to queueing delay reduction. Rather, it helps indirectly through assuring individual flows their reserved resources.

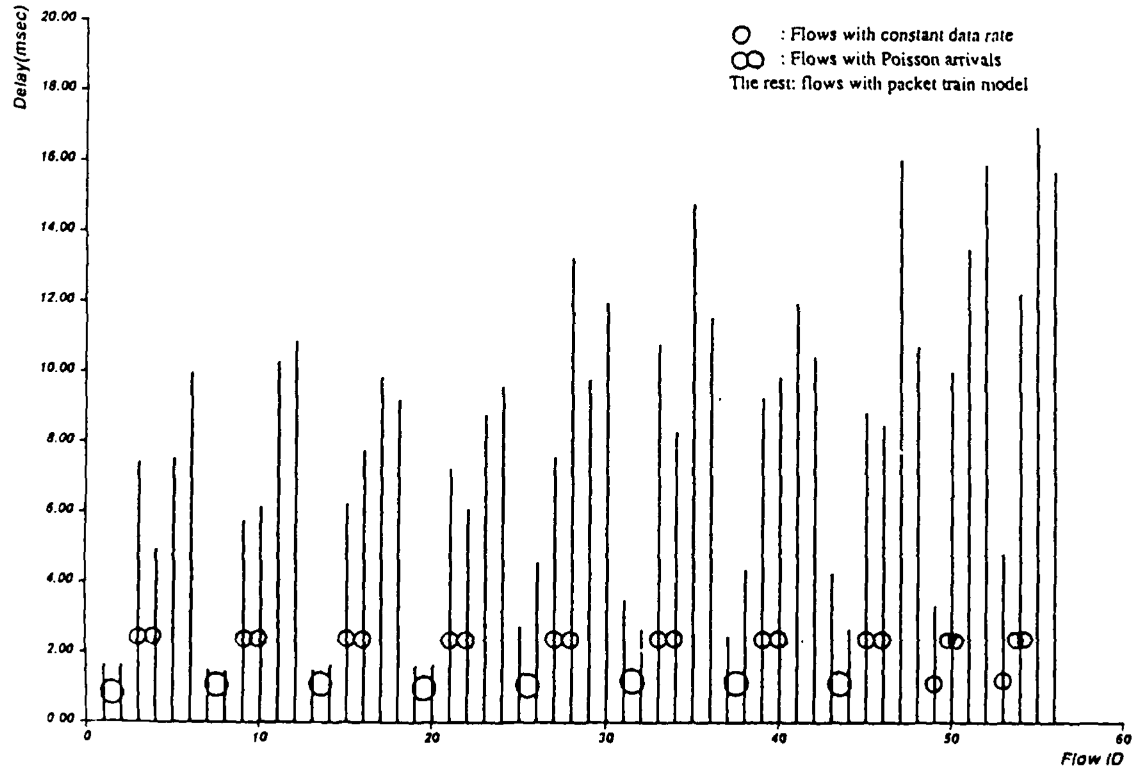
Queueing Delay with Different Data Generation Patterns *Although statistical* multiplexing absorbs certain randomness and burstiness in individual flows' data transmission, highly bursty data arrivals still increase queueing delay.

Because of the strict service ordering enforced by the VirtualClock mechanism, simulation shows that a higher burstiness in a flow's data generation is reflected back to a higher variance in that flow's transmission delay. To see this, we ran a simulation with three different data generation models, constant rate, Poisson arrival, and packet-train. The packet-train model has the highest burstiness among the three. The test condition is the same as that in Test-FP-One except that the last four flows were removed to lower the channel utilization[8] (the measured utilization is 78%), and that for flows 1 $\sim$ 48, the data generation model is changed to two constant-rate, two Poisson arrival, and two packet-train in a repeated pattern. Flows 49 $\sim$ 56 repeat the pattern of one constant-rate, one Poisson, and two packet train. The average and deviation of the queueing delays of the flows is shown in Figure 4-17. The numerical values are presented below.

[8]Experiments show that when the utilization is high, say above 80%, the difference of the queueing delay among different data source models gradually diminishes.



Average end-to-end queueing delay of flows with diverse data sources



End-to-end queueing delay deviation of flows with diverse data sources

Figure 4-17: The average and the deviation of packet queueing delay with difference data generation models.

Queueing delay statistics of diverse data patterns

	Average Delay (msec)			Delay Deviation		
	1-hop	2-hop	3-hop	1-hop	2-hop	3-hop
Constant rate	1.76	4.50	5.57	1.58	3.40	4.09
Poisson arrival	4.55	8.37	11.40	6.44	9.74	11.15
Packet train	6.49	10.47	15.25	9.50	12.17	15.49

The flows with the train data generation received the highest variance in transmission delay. It seems fair to let bursty data sources bear the result of their own behavior. In particular, notice that the flows with the Poisson arrival model have both a lower average delay and a smaller deviation than the flows with the packet-train model.

Effect of Priority

In principle, high priority flows may enjoy a low utilization as if low priority traffic did not exist. Without pre-emption, however, low priority traffic does have an effect on the delay of high priority flows: on average the latter will experience a half packet delay from the former at each non-idle queue.

Real-time applications, which require a transmission latency bound, may make up a large portion of the total network traffic in the near future. Therefore we are interested in identifying the effect of priority in queueing delay reduction when a large portion of the load may desire a high transmission priority. For this purpose, we ran another simulation test with the same condition as Test-FP-One, except that, in each path length group, half of the flows were given a higher priority than the other half. The average queueing delay of the flows are summarized below. Comparing with the results from Test-FP-One, the priority flows received a factor of $2.5 \sim 3$ reduction in the average queueing delay.

Average Queueing Delay (msec)

	utilization	1-hop	2-hop	3-hop
Priority flow (50% of load)	0.86	2.97	5.93	8.88
Non-priority flow (50% of load)		13.33	23.78	33.57
Test-FP-One	0.86	7.76	14.58	22.37

To get a qualitative estimate from an analytical model, we consider the average waiting time of an M/G/1 queue, W_ρ (see [3], pp.143)

$$W_\rho = \frac{R}{1-\rho}$$

where ρ is the channel utilization, and R the mean residual service time. If the traffic is divided into two different priority classes, the average waiting time in queue for each class is given by (see [3], pp.160)

$$W_{\rho_1} = \frac{R}{1-\rho_1}, \quad W_{\rho_2} = \frac{R}{(1-\rho_1)(1-\rho_1-\rho_2)}$$

where ρ_i is the channel utilization for priority i. Priority-1 users have a queueing delay reduction by a factor of

$$\alpha = \frac{W_\rho}{W_{\rho_1}} = \frac{1-\rho_1}{1-\rho} \qquad (4.1)$$

Substituting ρ by 86% and ρ_1 by $\rho/2$, respectively, we get $\alpha \approx 4$, roughly in agreement with the simulation results. If we substitute ρ by some other values, $\rho = 0.80$ would lead to $\alpha = 3$, and $\rho = 0.90$ would give α a value of 5.5.

In a low bandwidth and high utilization environment, a priority mechanism may give a significant queueing delay reduction measured in time. For instance, assuming an average queue length of 5, for a channel bandwidth of 50 Kbps and a packet size of 250 bytes, transmission time of one packet is $T = 40$ msec. A factor of 5 reduction brings the average queueing delay down from 200 msec to 40 msec. But if the channel bandwidth is 1 Gbps, T will be 2 μsec, and a factor of 5 reduction is only 8 μsec. The effect of a priority mechanism in reducing transmission delay is much less significant in high-speed networks. It might be a different story, however, if only a small percentage of the load is given priority.

Effect of Channel Utilization

Under statistical multiplexing, the channel utilization has a major impact on the packet queueing delay. Under the given packet train model, we would like to get some empirical measure on how the queueing delay varies with the channel utilization.

We performed four simulation runs, each with a different channel utilization, to observe the relation between the utilization and the queueing delay. The average and the 99th percentile of the queueing delay of the flows are summarized below, together with the result from Test-FP-One (the last line in the table).

Average queueing delay under different channel utilization

Channel	1-hop		2-hop		3-hop	
Utilization	Delay(msec)	99-tile	Delay(msec)	99-tile	Delay(msec)	99-tile
0.67	3.20	25.3	4.98	31.6	7.18	38.1
0.72	3.69	28.8	6.29	38.9	9.35	49.9
0.77	4.54	35.4	7.52	48.4	11.98	68.8
0.82	6.08	51.7	9.69	68.8	15.16	101.6
0.86	7.76	86.1	14.58	151.5	22.37	209.8

When the utilization increased from 67% to 77%, the average delay was increased by about 50%. But when the utilization increased from 77% to 86%, the average delay was nearly doubled. Also notice that at high utilization, the 99th percentile of the queueing delay increased much faster than the average, suggesting that adjusting the channel utilization can be an effective measure to meet latency bound requirements.

Phase-Locking

Recall that with the packet-train model, packets in one burst are transmitted with a constant burst rate. It is conceivable that some packets in a long burst may momentarily get phase-locked with packets from other flows. The moving-average control at flow sources may also cause phase-locking.

Transient phase-locking was observed indirectly from end-to-end packet delay samples. Figure 4-18 shows a 20-second sample of the end-to-end packet delay of a 1-hop flow in Test-FP-One. The narrow flat areas on the curve indicate that more than one packet experienced exactly the same transmission delay. If this delay is longer than the minimum transmission time, it implies that the packets experienced exactly the same queueing delay, i.e. the packets were momentarily phased-locked with packets from other flows.

Phase-locking does not seem to endure or cause significant queueing delay increase. In the test run of diverse data generation model, one-third of the flows transmitted at constant-rate, the most likely condition of getting permanently phase-locked with each other. Nevertheless, those constant-rate flows received a much lower end-to-end queueing delay, measured in both average and variation, than other random flows.

There are many random factors that influence the network traffic (as long as not all users are transmitting at a constant rate), and we have never observed persistent phase-locking in

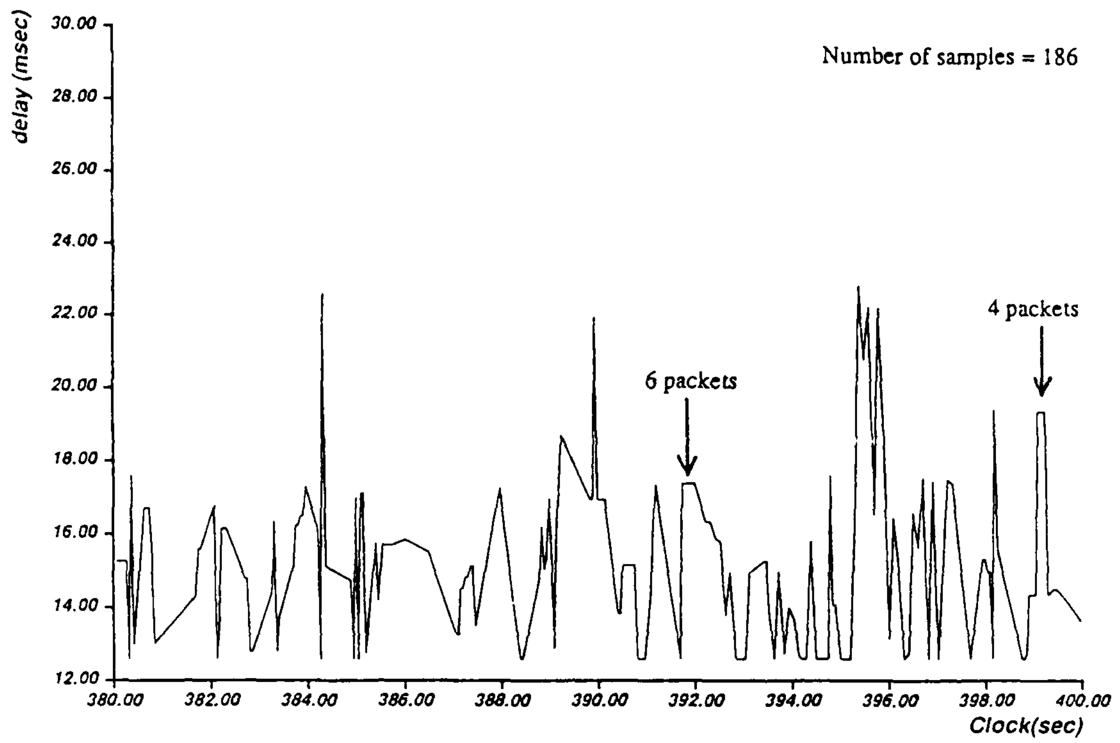


Figure 4-18: Samples of end-to-end packet delay showing transient phase-locking.

simulation.

Summary of Latency Control

Several mechanisms in our control algorithm have an effect on transmission latency; here we clarify the specific role each one plays:

- The goal of network control, resource reservation and usage enforcement, is to allocate adequate resources to individual flows to assure the service quality. Without this, congestion may occur and cause data losses.

  In addition to preventing congestion, network control can also adjust the utilization to reduce network queueing delay.

- As a control enforcement, VirtualClock builds firewalls among flows to guarantee the reserved resources for individual flows.

  As a side-effect of the VirtualClock mechanism, burstier flows experience a larger queueing delay variance.

- The user behavior envelope constraint, the moving-average control at the flow source, cuts off peaks in data generation and therefore reduces the long tails in the queueing delay distribution.

  One of the flow throughput parameters, AI, can be adjusted to reduce the queueing delay variation; the tradeoff is a tighter constraint on the data transmission.

- A priority mechanism can reduce the average queueing delay for a portion of the load by some factor α (as given in Equation 4.1).

We expect that the orders of magnitude channel bandwidth increase brought forth by fiber optics will have the most significant effect on queueing delay reduction. Tight latency bounds required by many real-time applications, which are difficult or maybe even impossible to achieve in today's low-speed networks, may be easily met in a high-speed network.

4.5.2 Performance of Short Transfers

As we discussed in Chapter 3, short transfers in a Flow Network are treated as a special flow and given a low priority. Simulation tests show that the average delay of short transfers is determined by the network utilization.

In simulation, short transfer requests are generated as Poisson arrivals, the length of each request is an exponentially distributed random variable with a mean of 3 packets. The source and destination address pair is randomly chosen from all the combinations of the host pairs that cross at least one switch-to-switch link. All the packets in one short transfer are sent in a burst (i.e. with zero inter-packet delay). The total volume of short transfers can be adjusted by the number of requests generated per unit time.

Three simulation runs were performed to examine the performance of short transfers in a Flow Network. The first one was arranged in the same way as Test-FP-One except that short transfers were added. The second test was the same as the first one except that flows 57 $\sim$ 60 were removed to reduce the channel utilization. And the third test was the same as the second except that the volume of short transfers was increased by a factor of 5. Each test simulated a 10 minute run of the real system.

The measured average queueing delay from the three tests are given below. U_{total} is the measured channel utilization, U_{flow} the utilization by regular flows, and U_{short} the utilization by short transfers. The queueing delay of short transfers is computed from the measured end-to-end delay by subtracting the transmission and propagation delay of a 2-hop path.

Average queueing delay of flows and short transfers (in msec)

	U_{total}	U_{flow}	U_{short}	Normal Flow			Short Transfer	
				1-hop	2-hop	3-hop		loss
Test-1	90%	86%	4%	8.75	15.82	23.11	256.85	0
Test-2	80%	76%	4%	4.42	8.31	13.15	80.07	0
Test-3	95%	76%	19%	5.76	11.94	15.06	462.80	0.3%

We see that when the channel utilization is moderate and the the total volume of short transfers is low (Test-2), the delay of short transfers is acceptable. When the utilization is high (Test-1), the delay of short transfers increases sharply. As the volume of short transfers increases, the network may or may not have enough spare capacity to accommodate them. If the possible region of the volume of short transfers can be estimated in advance, one can adjust the control parameter, $U_{capacity}$, to leave adequate resources for short transfers. If such an estimate is not available, or if some users abuse the Flow-0 for large quantities of data, short transfers may suffer long delays or even losses.

We also see that regular flows are well isolated from short transfers. When the utilization approaches 1, there is a slight increase in the flows' queueing delay. This is because the channels are always busy, and packets from regular flows are delayed by packets from short transfers that are already under transmission.

4.5.3 Control Stability with Dynamic Load

The Flow Network is also tested under dynamic load (i.e. randomly generated flow requests, as described in Section 4.3.2). It is rather difficult to quantify the network performance. Averaging the channel utilization over an entire simulation run becomes meaningless because there are periods when demand is low.

The strongest claim we can make here is that congestion never occurred; for all the flows accepted, the network meets their average throughput and delay requirements. In case of resource shortage, new requests are rejected; the network signals rejected flows when resources become available. The traffic stability comes from enforcing a reservation control.

4.5.4 Effect of Network Channel Heterogeneity

Now consider the effect of network channel heterogeneity measured in delay and bandwidth. It causes serious performance difficulties in today's packet switching networks that use the window flow control either at the network or at the end-to-end level. Propagation delay variations of different paths make window size selection difficult. Bandwidth mismatch leads to high demand for buffer space at bottleneck points, which may result in both long queueing delays and data losses.

With a transmission rate control, the above problems are largely avoided. Because data transmissions are regulated by time, the throughput of a flow is no longer sensitive to the RTT value. With resource reservations, the traffic passing bottleneck points is not allowed to exceed the capacity. And because the channel bandwidth is one of the parameters used in computing flows' average interval bound, flows passing a low-speed channel are constrained by both a low average rate and a small average interval. The rate-based flow control can maintain roughly identical packet queue distributions for all the channels; the effect of the channel heterogeneity is reduced to the transmission, and hence the queueing, delay.

We performed simulation tests over a heterogeneous network topology model as shown in Figure 4-19. There are 14 switches and 30 heterogeneous links. Two of the links have the

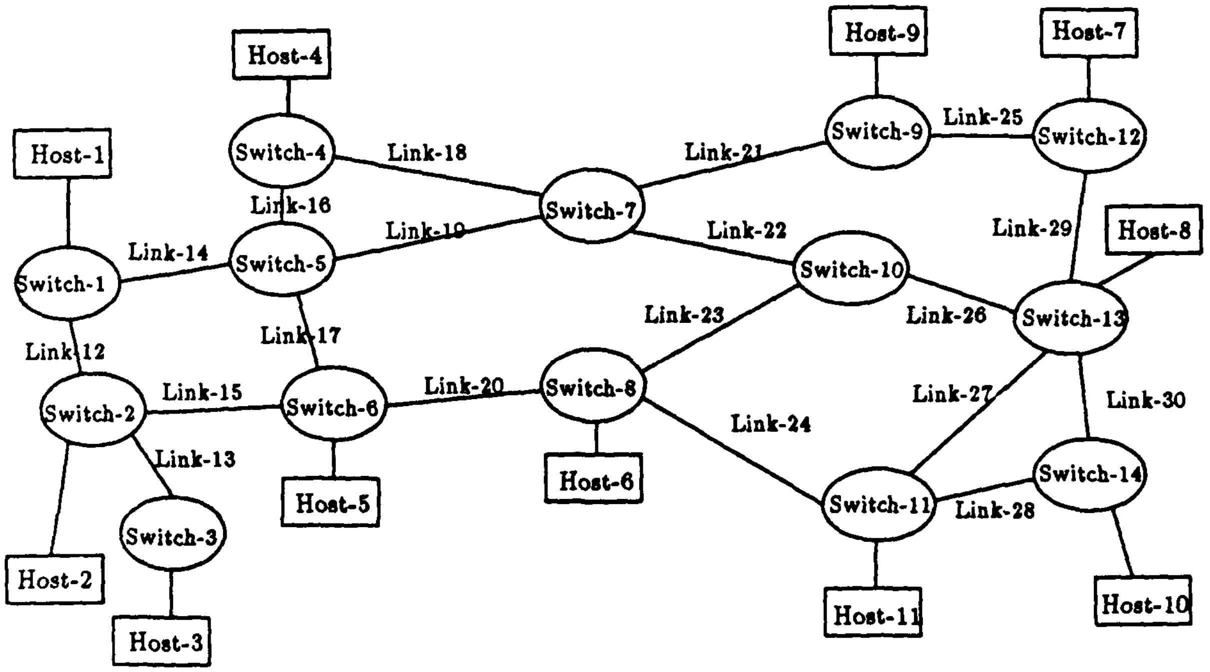


Figure 4-19: A heterogeneous network topology model.

propagation delay of a satellite channel, 250 msec, and the others 5 to 10 msec. Because of the computation speed restriction, a moderate bandwidth region from 50 Kbps to 2 Mbps is chosen.

Because of the complexity of the topology, tests performed so far are still very preliminary. One result we can present is that, buffer demands at switches connecting to low-speed channels are about the same as at other places. The demand on buffer space is mainly a function of $U_{capacity}$ and the average interval bound, largely independent of the channel bandwidth. Below is the measurement of two channels over a two minute simulation run. The channels are under (more or less) the same utilization but with different bandwidths. Figure 4-20 presents the queueing samples of the two channels.

Channel measurement in a heterogeneous network

Channel	Bandwidth	Utilization		Queue Length		
		mean	dev	mean	dev	99-tile
12	2 Mbps	0.80	0.06	2.26	1.14	6
14	50 Kbps	0.79	0.32	2.70	1.57	7

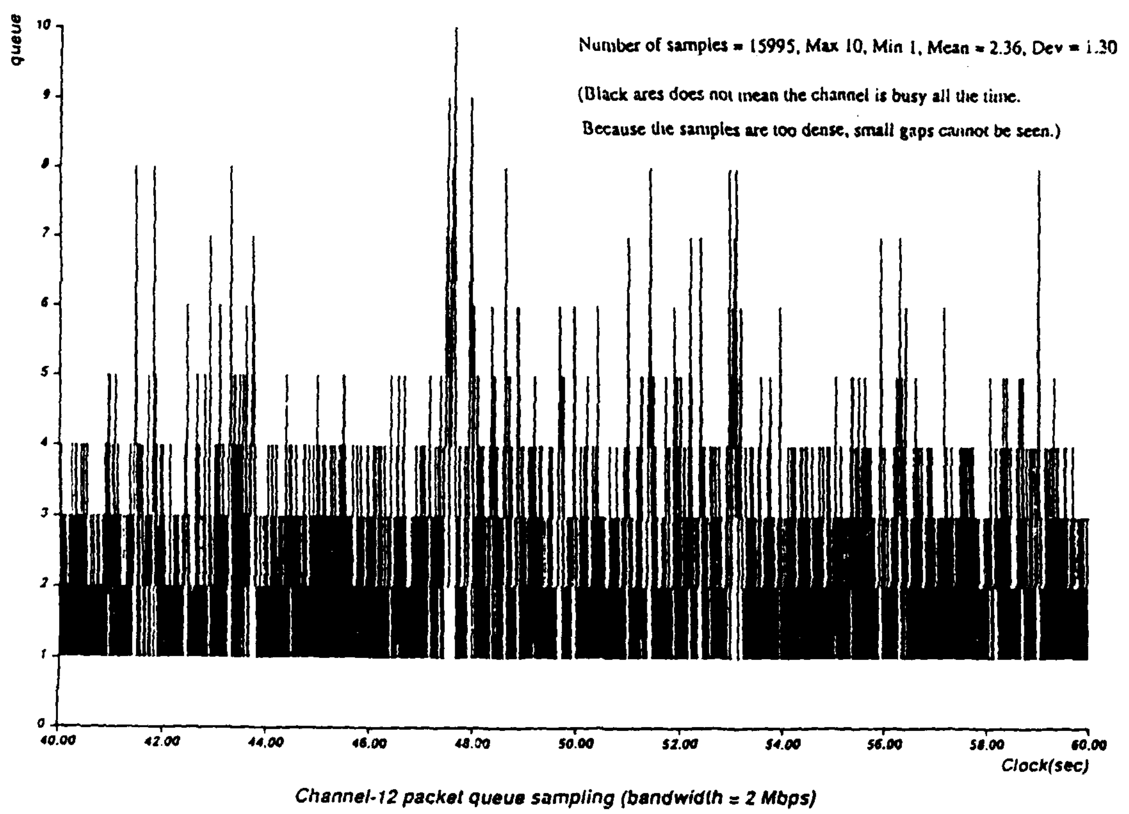
4.6 Summary

Simulation results presented in this chapter show that the Flow Network can ensure performance in the following respects:

1. Congestion is eliminated even in the face of excessive input requests.
2. The network load is stable.
3. Average throughput requirements of flows can be met, even in the presence of misbehaving users. The service is stable despite dynamic load changes.
4. Average delay requirements can be met with the given data generation model.

Maintaining a proper utilization level is the core of network delay control, which is assured by our network reservation and rate-based flow control mechanisms. It is difficult for individual end users to adjust network utilization. It is also difficult to use a window mechanism to control the utilization, because it tends to drive the network to a full utilization when the demand is sufficiently high.

In addition to testing the Flow Network design, we also simulated TCP-SS, an end-point control and window-based protocol architecture. The results show agreement with our earlier analysis about the drawbacks of end-point control and window mechanisms. A proposed enhancement to stateless network control, Random-Drop, was also simulated. The results show



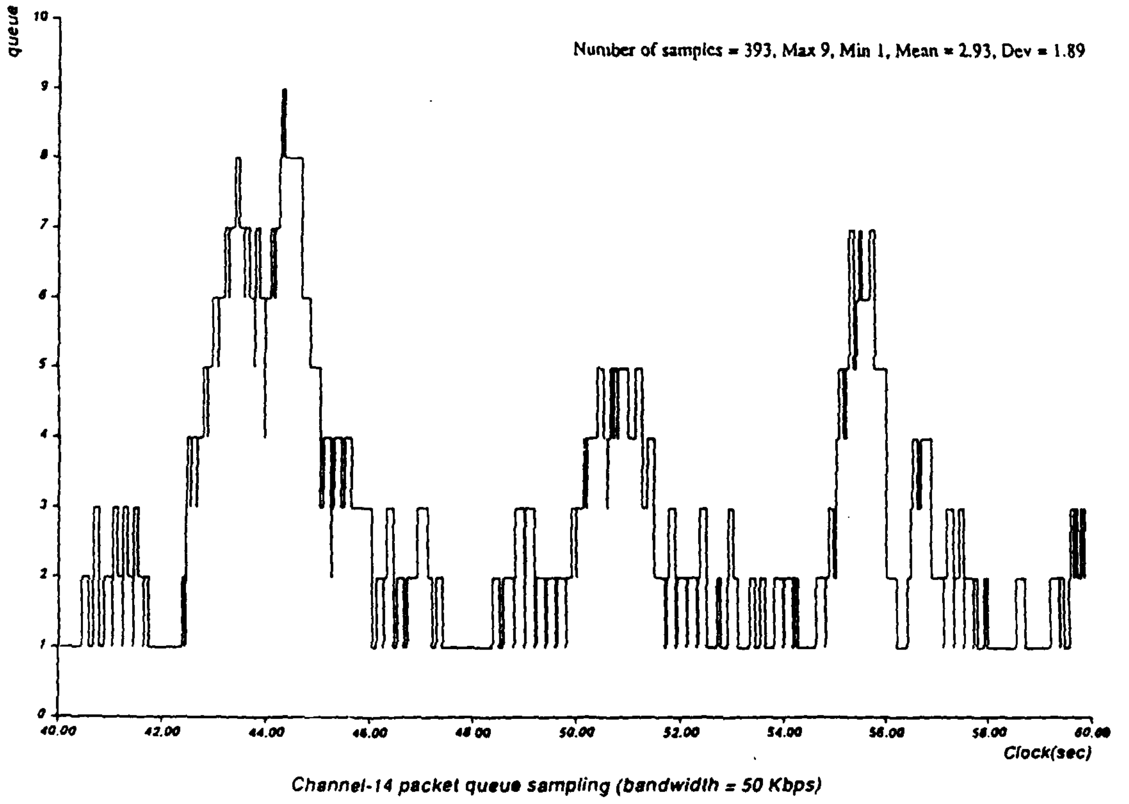


Figure 4-20: Packet queue samples of two channels with different bandwidths.

that, with TCP-SS as the end-to-end protocol, Random-Drop does not prevent misbehaving users from taking an unfair share of network resources.

Experience shows that simulation is a viable approach to test our design validity. Simulation has served as an invaluable aid throughout the design and testing process.

Chapter 5

Conclusions and Future Work

In this final chapter, we first summarize the contributions of this research to packet switching network design, and then discuss possible extensions of the results, followed by a list of remaining work for future study.

5.1 Summary of the Contributions

This dissertation study consists of two major parts: high level design decisions, and the design of a framework of rate-based traffic control protocols for packet switching networks. The design was tested through simulation.

5.1.1 Architectural Design Issues

The thesis of this dissertation is a three-fold argument supported by simulation results: packet switching networks should have performance control built into the network; the network control should employ rate-based mechanisms; and assurance of service quality should be based on resource reservation. We also propose a concept of a *user behavior envelope* that each flow should obey.

Network Control

The goal of network traffic control is to allocate adequate resources to individual flows to assure the service quality. The discussion in Chapter 2 concludes that a stateless network may not be able to achieve this goal. The simulation results presented in Chapter 4 suggest that when a network is unable to discriminate against misbehaving users, it cannot assure a satisfactory service.

In this thesis, I propose a reservation-based network control system. One concern about such an approach is service robustness, since network control requires maintaining traffic state inside the network. Chapter 2 analyzed necessary conditions for a robust system, and concluded that, if the network has no fate-sharing among switches, it can offer the same, or even higher, degree of robustness in service than a datagram network, because it can prevent congestion which often makes a datagram network unusable. This speculation is yet to be verified.

Transmission Rate Control

Network utilization is maintained through controlling the transmission rate over all flows. Rate-based flow control is a better choice than the conventional window mechanisms, because the network transmission capacity is measured in rate, because window mechanisms tend to drive the network into a high utilization, which may lead to high queueing delays, and because the average transmission rate of a window-controlled flow is determined by $(Window/RTT)$; the association with the RTT raises difficulties as we already explained in Chapter 2.

One difficulty in packet traffic control has been coping with unknown and bursty traffic load. We take the approach of first letting users specify the throughput rate and then proposing a user behavior envelope based on their throughput declaration.

Resource Reservation

The motivation for building a reservation-based control system is to avoid the instability that can be introduced by feedback control delay. The minimum delay between a change in the traffic, issuing a control action accordingly, and observing the control effect is a measurement delay plus one round-trip-time. During this time period, the control and the load are "out-of-phase". The higher the network speed, the more packets may be transmitted during the out-of-phase period.

Because of the control delay, it is no longer feasible for high-speed networks to use feedback control at the data transmission level. The Flow Network enforces resource reservations in order to assure service quality, and regulates the network load at the reservation level.

User Interface

We proposed a service specification interface to provide a means for applications to express their requirements. These service requirements are then interpreted as a user behavior envelope to constrain data transmissions within the claimed enclosure. Users get to choose and get to obey whatever they choose as well.

Although the concept of a user behavior envelope was introduced as a solution to the problem of flow measurement, we believe it is a mandatory part of rate-based flow control in general. Because packet switching offers unbounded flexibility to users, a clearly defined user behavior envelope is needed to counter-balance the flexibility. In fact, window flow control presents a well defined transmission constraint. Setting constraints on users is a necessary cost, which ought to be recognized explicitly and loudly. Much work needs to be done on how to design application protocols that can adjust themselves to the constraints.

5.1.2 A Framework for Rate-Control Network Architecture

This research contributed a framework for a rate-based network control system. There are two major components in the architecture: a user model, and a network control algorithm.

The user model, the *flow*, is associated with service quality requirements. A flow threads through elements of network resources to make the reservation. The claimed service requirement is also used as its own behavior envelope to constrain the data transmission.

The network enforces a reservation-based traffic control to maintain the utilization, and, by the VirtualClock mechanism, assures all the running flows the allocated resources.

5.1.3 Technical Issues

This research contributed solutions to a number of problems raised by rate-based flow control.

Rate Control Parameters

This research introduces a new concept, *average interval*, and uses two parameters, average rate (AR) and average interval (AI), to describe and control random data flows.

The average interval defines a checking point in flow measurement and sets a bound on transmission variation. We let each flow make its own AI choice, although the value is necessarily bounded by the network.

The burstiness degree (the ratio of peak to average rate) may also be an important parameter in transmission rate control. A constant value of 2 has been used in our simulation experiments. The effect of different burstiness values will be addressed in a future study.

As a final remark, the concept of the *average interval* seems particularly inspiring.

$$\frac{1}{average\ rate} < average\ interval(AI) < flow\ duration$$

Tuning the value of AI to the left limit, we would get a TDM system; tuning the value to the right limit (assuming the duration can be known beforehand), we would set no constraint on the flow's transmission. AI provides us a useful tuning knob between the system constraints and the service flexibility. Use of AI is a promising direction to be further pursued.

User Behavior Envelope

All control algorithms imply constraints on users. In the Flow Network, such constraints are communicated to the end user as a user behavior envelope. The Flow Network defines the user behavior envelope to be the transmission of no more than $(AR \times AI)$ data over every average interval.

In our simulation tests, the user behavior envelope was implemented by a moving-average algorithm. Depending on the characteristics of individual applications, other means may be more viable. For example, a bulk data transfer application can easily schedule the data fetch from the storage at a specific rate, with no further constraint at the transmission point.

VirtualClock Mechanism

An important contribution of this research is the development of the VirtualClock mechanism. The role of VirtualClock in traffic control is two-fold: it is a meter that measures the average rate of a statistical flow; it also performs an average-rate enforcement in data forwarding.

Depressing Synchronized Actions in Distributed Control

In the course of testing the Flow Network design, we re-discovered a common issue in distributed control systems that we call synchronized actions. We explained the cause and suggested using randomization to reduce synchronization in distributed control.

5.2 Future Applications of the Results

5.2.1 Assisting Other Network Functions

Network access control and accounting are among the major functions an operational network needs to provide. Both can be easily added to the Flow Network. Access control can be enforced as part of flow setup. User identification and authentication information may be added into the Flow-Request message, which may then be checked by the network. If the request is approved, an encryption key can be carried in the Request-Reply message back to the flow source; all packets from this authorized flow can then be authenticated.

After user identity is verified, the network will be able to perform accounting on a per-flow basis. Because the average rate is not only given but also enforced, the accounting function needs only to measure the duration of each flow, instead of counting individual packets.

5.2.2 Extending Flows to Inter-Networking

Here we consider how to extend the Flow Network to an internet environment, where a number of networks are interconnected.

Inter-Connecting Flow Networks

If all the participating networks support the concept of flow, the inter-connection would be easy. Following the telephone networks' approach, where connections remain unchanged no matter which level of the network hierarchy a call may go through, flows can be concatenated across network boundaries. One issue that must be resolved is flow ID assignment, because different networks may have overlapped flow ID spaces.

Inter-Connecting Heterogeneous Networks

If the networks to be inter-connected have different architectures, including datagram networks and VC networks, as well as Flow networks, meeting the minimum functionality of passing packets through would still be easily achievable. At the interface between a Flow Network and a VC network, a virtual connection can be set up for each flow and vice versa. At the interface between a Flow Network and a datagram network, packets passing to the datagram network can be forwarded as independent datagrams. Packets from the datagram network to the flow network need be identified by certain attributes, such as source and destination addresses, so that the Flow Network can, as much as possible, set up flows accordingly, and forward the rest through Flow-0. Although the flows identified by the source-destination addresses have no performance specification, they nevertheless allow the network to have a better knowledge of the traffic than facing individual datagrams.

The throughput and delay assurance of the transmission, however, would not be achievable. Network performance has to be supported bottom-up. Because neither the datagram network nor the VC network assures transmission throughput, an inter-connection on top of them cannot achieve it. The performance of a flow would then largely be up to the network load at the particular time of transmission.

5.3 Future Research Issues

This dissertation is only an initial attack on rate-controlled packet switching networks; substantial issues remain to be resolved. A partial list of the issues that can be clearly identified at this time follows.

5.3.1 Resource Overbooking

When a bursty flow specifies the AR value as its expected average throughput, because of the constraint of the user behavior envelope, the actual throughput may be lower than the expected average, hence the resources are overbooked. Such overbooking can become negligible if the average interval is chosen adequately large, although a large average interval would stress the network in terms of the buffer capacity and queueing delay.

On the other hand, flows may also intentionally overbook resources for a better service. There may be different views on such intentional overbooking. One view may consider that the overbooking is legitimate (especially if the user pays for the booked resources) and networks should be well engineered to satisfy users' requests. Another view may propose to measure the overbooking and allocate the overbooked resources to other users. VirtualClock can be used to measure a flow's overbooked resources in the following way: each time a flow is checked, the switch may record the difference between the real time and the flow's VirtualClock. The amount of overbooking can be estimated from the sum of the difference over a number of AI periods. However, it seems risky to take back part of the resources the network has promised to users.

How much performance improvement can a flow gain by a slight overbooking? This seems to be an interesting question to look into.

5.3.2 Latency Control

It is highly desirable to express, either analytically or empirically, the queueing delay as a function of the average train length, the burstiness degree, and the average interval bound set by each channel, in order to estimate the packet loss ratio under a given latency bound.

In particular, we would like to first investigate the impact of the burstiness degree on the packet queue distribution. The main results presented in Chapter 4 were obtained with a traffic burstiness degree of 2. Secondly, we would like to look into the relation between the average interval bound and the delay distribution. Finally, we would also like to experiment with adjusting the channel utilization measurement to maintain a proper queueing delay level, as discussed in Section 3.7.3.

5.3.3 Control Dynamics

In this thesis, we discussed the delay effect of feedback control. Without the knowledge of traffic dynamics, our design has carefully avoided dynamic traffic adjustment in favor of control stability. The next step is to add dynamic traffic adjustment to the Flow Network. In particular, we would like to be able to increase the service availability to high priority applications by interrupting or slowing down low priority flows when the network is in momentary resource shortage.

We would experiment with adjusting the value of the control parameter U_{low} to understand the relation between maintaining a high utilization and maintaining control stability. In fact U_{low} plays a similar role to the utilization control parameter in the Mosely algorithm, except that U_{low} controls the load at a different level — it controls the dynamic adjustment of flow reservations, instead of flows' instantaneous transmission rate.

A low U_{low} value will lead to a stabler control, by trading off the utilization (because now flows may not be signaled promptly after resources become available). One possible experiment is to combine a high value of U_{low} with our *randomization* technique: when the channel utilization is below U_{low}, the switch may signal rejected flows with a given probability.

5.3.4 Application Protocol Design

This research proposed two parameters, average rate and average interval, to be used to control data transmission. Application protocols need be designed accordingly. Given an application, how does one choose the values of AR and AI, especially AI? In addition, because the maximum AI value is bounded by the network, how will the application protocol adjust to the constrained AI value without affecting the user's perceived performance? These questions must be answered before the proposed control algorithm can be implemented in real networks.

In addition to answering the above question for specific applications, can we provide general guidelines to application protocol designs? Much research effort is needed to answer this question.

5.3.5 Relation with Routing

Finally, the relationship with routing is an important yet difficult issue. This research assumes the availability of a routing server. When a user wants to transmit, it sends a flow request along the path provided by the routing server.

Traffic control and routing interact with each other. Much work has been done in the area of making routing decisions based on dynamic network load conditions. We feel that the role of routing and traffic control should be separated into different stages in carrying out a transmission request. A routing decision is concerned not only with resource availability, but also with usage policy and accounting issues. Assuming that the network is well engineered, route selection can be mainly based on topology information (which includes channel bandwidth and delay features) with adaptivity to network component failures. It may also be feasible to integrate long term load information, such as different traffic rush hours due to different time zones, into routing decisions.

The robustness of the Flow Network is also mainly a routing problem. After a switch failure is detected, the routing service must be able to promptly offer an alternative path, if one exists, to repair broken flows.

This research is both an end and a start. It presents a framework for a rate-based network control system. It also brings up more open issues than it resolves. Those issues and experiments are necessarily left for future study, however, to allow this dissertation to be concluded.

Bibliography

[1] OSI 8073: *Information Processing Systems - Open Systems Interconnection - Connection Oriented Transport Protocol Specification.* International Organization for Standardization. July 1986. (Ref.no. ISO 8073-1986 (E))

[2] A. Demers, S. Keshav, and S. Shenker. "Analysis and Simulation of a Fair Queueing Algorithm". March 1989. To appear in SIGCOMM'89 proceedings.

[3] D. Bertsekas and R. Gallager. *Data Networks.* Prentice-Hall, 1987.

[4] P. Buning and R. Steger. "Graphics and Flow Visualization in Computational Fluid Dynamics". In *Proceedings of the 7th Computational Fluid Dynamics Conference*, AIAA, July 15-17 1985.

[5] V. Cerf. "Packet Communication Technology". In *Protocols and Techniques for Data Communication Networks*, Chapter 1, pages 1–34. Prentice-Hall, 1981.

[6] D. Chiu and R. Jain. "Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks". 1989. To appear in *Journal of Computer Networks and ISDN.*

[7] K. Christian. *The UNIX Operating System.* John Wiley and Sons, 1983.

[8] D. Clark. "The Design Philosophy of the DARPA Internet Protocols". In *Proceedings of Symposium on Communication Architectures and Protocols*, ACM SIGCOMM, August 1988.

[9] D. Clark, M. Lambert, and L. Zhang. "NETBLT: A Bulk Data Transfer Protocol". RFC-998, Network Information Center, SRI International, March 1987.

[10] A. Drake. *Fundamentals of Applied Probability Theory.* McGraw-Hill Book Company, 1967.

[11] K. Seo et al. "Distributed Testing and Measurement Across the Atlantic Packet Satellite Network (SATNET)". In *Proceedings of Symposium on Communication Architectures and Protocols*, ACM SIGCOMM, August 1988.

[12] W. Feller. *An Introduction to Probability Theory and Its Applications.* John Wiley & Sons, 1968.

[13] M. Fine and F. Tobagi. "Packet Voice on a Local Area Network with Round Robin Service". *IEEE Transactions on Communications*, COM-34(9):906–915, September 1986.

[14] F. George and G. Young. "SNA Flow Control: Architecture and Implementation". *IBM System Journal*, 21(2):179–210, 1982.

[15] M. Gerla and L. Kleinrock. "Flow Control Protocols", In *Computer Network Architecture and Protocols*, Chapter 13, pages 361–412. Plenum Press, 1982.

[16] B. Leiner, editor, Gigabit Working Group. "Critical Issues in High Bandwidth Networking". RFC-1077, Network Information Center, SRI International, November 1988.

[17] H. Frank, R. Kahn, and L. Kleinrock. "Computer Communication Network Design – Experience with Theory and Practice". In *Proceedings of National Computer Conference*, AFIPS Press, 1972.

[18] E. Hahne. *Round Robin Scheduling for Fair Flow Control in Data Communication Networks.* PhD thesis, Massachusetts Institute of Technology, December 1986.

[19] E. Hashem. Analysis of Random Drop for Gateway Congestion Control. M.S. thesis in progress, Massachusetts Institute of Technology, 1989.

[20] A. Heybey. *Rate-Based Congestion Control in Networks with Smart Links.* B.S. thesis, Massachusetts Institute of Technology, 1988.

[21] *IEEE Journal on Selected Areas in Communications.* December 1988. A special issue on Broadband Packet Communications.

[22] *IEEE Journal on Selected Areas in Communications.* October 1987. A special issue on Switching Systems for Broadband Networks.

[23] J. McQuillan, I. Richer, and E. Rosen. "The New Routing Algorithm for the ARPANET". *IEEE Transactions on Communications*, COM-28(5):711–719, May 1980.

[24] V. Jacobson. Presentation at the Internet Engineering Task Force meeting, April, 1987.

[25] V. Jacobson. "Congestion Avoidance and Control". In *Proceedings of Symposium on Communication Architectures and Protocols*, ACM SIGCOMM, August 1988.

[26] R. Jain and S. Routhier. "Packet Trains - Measurements and a New Model For Computer Network Traffic". *IEEE Journal on Selected Areas in Communications*, SAC-4(6):986–995, September 1986.

[27] R. Jain. *Divergence of Timeout Algorithms for Packet Retransmissions.* Technical Report 329, Digital Equipment Corporation, 1985.

[28] R. Jain and K. Ramakrishnan. "Congestion Avoidance in Computer Networks with a Connectionless Network Layer: Concepts, Goals, and Methodology". In *Proceedings of Computer Networking Symposium*, 1988.

[29] D. Karvelas and A. Leon-Garcia. "Performance of Integrated Packet Voice/Data Token-Passing Rings". *IEEE Journal on Selected Area of Communications*, SAC-4(6):823–832, September 1986.

[30] 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 DEC-TR-510, Digital Equipment Corporation, 1987.

[31] L. Kleinrock. "A Decade of Network Development". *Journal of Telecommunication Networks*, Spring 1982.

[32] M. Lambert. "An End-Point Adaptive Rate Control Strategy for the NETBLT Protocol". Unpublished research notes, MIT Laboratory for Computer Science, May 1988.

[33] B. Leiner. "Network Requirements for Scientific Research". RFC-1017, Network Information Center, SRI International, August 1987.

[34] M. Gardner, I. Loobeek, and S. Cohn. "Type-of-Service Routing with Loadsharing". In *Proceedings of GLOBCOM*, November 1987.

[35] TCP-IP mailing list. The TCP-IP mailing list is a special-interest-group mailing list moderated by the Network Information Center (NIC) located at SRI. In TCP-IP mail discussion, there have been numerous observations of malfunctioning hosts in the ARPA Internet.

[36] J. Mosely. *Asynchronous Distributed Flow Control Algorithms.* PhD thesis, Massachusetts Institute of Technology, October 1984.

[37] U. Mukherji. *A Schedule-Based Approach for Flow-Control in Data Communication Networks.* PhD thesis. Massachusetts Institute of Technology, February 1986.

[38] J. Nagle. "Congestion Control in TCP/IP Internetworks". *ACM Computer Communications Review*, 14(4), October 1984.

[39] Performance and Congestion Control Working Group, Internet Engineering Task Force. "Gateway Congestion Control Policies". Draft, January 1989.

[40] J. Postel. "DoD Standard Internet Protocol". RFC-791, Network Information Center, SRI International, September 1981.

[41] J. Postel. "DoD Standard Transmission Control Protocol". RFC-793, Network Information Center, SRI International, September 1981.

[42] R. Jain, K. Ramakrishnan, and D. Chiu. "Congestion Avoidance in Computer Networks with a Connectionless Network Layer". In *Innovations in Internetworking*, Artech House, 1988.

[43] K. Ramakrishnan and R. Jain. "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with Connectionless Network Layer". In *Proceedings of SIGCOMM'88*, 1988.

[44] J. Rinde. "Tymnet: An Alternative to Packet Technology". In *Proceedings of the 3rd International Conference on Computer Comuunications*, pages 268–273, August 1976.

[45] D. Sproule and F. Mellor. "Routing, Flow, and Congestion Control in the DATAPAC Network". *IEEE Transactions on Communications*, COM-29(4), April 1981.

[46] T. Sudu and T. Bradley. "Packetized Voice/Data Integrated Transmission on a Token Passing Ring Local Area Network". *IEEE Transactions on Communcations*, COM-37(3):238–244, March 1989.

[47] J. Turner. "New Directions in Communications (or Which Way to the Information Age?)". *IEEE Communications Magazine*, 24(10):8–15, October 1986.

[48] L. Tymes. "Routing and Flow Control in Tymnet". *IEEE Transactions on Communications*, COM-29(4):392–398, April 1981.

[49] L. Zhang. "Why TCP Timers Don't Work Well". In *Proceedings of Symposium on Communication Architectures and Protocols*, ACM SIGCOMM, August 1986.

[50] The best references to the ARPA Internet performance are discussions in the TCP-IP mail. Another source of information is the network measurement collected by BBN Communications Corporation.

OFFICIAL DISTRIBUTION LIST

Director Information Processing Techniques Office Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, VA 22209	2 copies
Office of Naval Research 800 North Quincy Street Arlington, VA 22217 Attn: Dr. R. Grafton, Code 433	2 copies
Director, Code 2627 Naval Research Laboratory Washington, DC 20375	6 copies
Defense Technical Information Center Cameron Station Alexandria, VA 22314	12 copies
National Science Foundation Office of Computing Activities 1800 G. Street, N.W. Washington, DC 20550 Attn: Program Director	2 copies
Dr. E.B. Royce, Code 38 Head, Research Department Naval Weapons Center China Lake, CA 93555	1 copy