Routing and Congestion Control in Datagram Networks

Zheng Wang

Submitted for the degree of PhD at

University College London

January 1992

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

ABSTRACT

Routing and congestion control can be viewed as two closely related processes in a resource control system which manages the network resources. One of the basic problems is to adapt to changing conditions. Our analysis shows that inadequate information and delayed feedback can cause oscillation and instability. In this dissertation we examine the problems in routing and congestion control in a datagram environment and propose a number of solutions. The essence of the ideas is to deal with momentary fluctuations and average steady trends separately.

The behavior of shorest-path routing algorithms is investigated. A new approach for dynamic routing based on the concept of decentralized decision making is introduced. A dynamic routing algorithm based on this new approach, *Shortest Path First with Emergency Exits (SPF-EE)*, is developed, which eliminates many of the problems that the Shortest Path First (SPF) algorithm exhibits under heavy and dynamic traffic. The SPF-EE algorithm allows local and temporary alteration of routes without global route updating. Simulation shows that the SPF-EE algorithm achieves substantial improvement over the SPF algorithm.

A new traffic adjustment algorithm called *synchronized traffic adjustment (STA)* and three information feedback schemes are proposed. The STA algorithm attempts to establish the sharing of the resources only when there are significant traffic changes and to maintain fairness by synchronized adjustment. Three congestion control schemes, based the STA approach and the proposed binary and quantitative feedback schemes, are presented. Extensive simulation shows that the STA algorithm eliminates the oscillation problems in the additive increase and multiplicative decrease (AIMD) algorithm and converges more quickly.

ACKNOWLEDGMENTS

I would like to thank my supervisor, Jon Crowcroft, for his guidance, help and encouragement during the course of my research. I would also like to thank the various members of the Computer Science Department for their assistance and advice during my stay at UCL: thanks are due to Peter Kirstein, Yuko Murayama, Dervish Deniz, Jiang Fan and Ping Hu.

I have benefited a great deal from the stimulating discussions in the TCP/IP community. Various people across the Internet have provided me valuable comments on a number of my papers where the ideas and preliminary results in this dissertation were first presented, my thanks and appreciation to Lixia Zhang (Xerox PARC), Keshav Srinivasan (AT&T Bell Labs), Paul Tsuchiya (Bellcore), Craig Partridge (BBN), Andrew Heybey (MIT), Paul McKenney (SRI), Bob Walsh (DEC), Van Jacobson (LBL), Allison Mankin (Mitre) and Gregory Finn (ISI).

I am also indebted to Jon Crowcroft, Ian Wakeman, Peter Kirstein and Soren Sorensen who have carefully read and commented on the draft version of this dissertation.

During my research I have been supported by the British Council and the State Education Committee of People's Republic of China for which I am grateful.

Finally, I would like to thank my parents and my wife Wei for their love and support during my study, and for making life more pleasurable on the good days and bearable on the not-so-good ones.

TABLE OF CONTENTS

LISTS OF FIGURES

LISTS OF TABLES

CHAPTER ONE

INTRODUCTION

The merging of computers and communications has had a profound influence on human society. Computer networks now play an essential role in the information technology and have become the key communication infrastructure that makes worldwide scientific, commercial and military coordination and operation possible. In the future, powerful personal computers and high speed networks together will provide us a magic window to the fantasy world of the Global Village and Virtual Reality. Wider use of electronic mail, video telephone, teleconferencing, telecommuting, teleshopping and telelearning etc may have revolutionary impacts on the way we live and work, and may eventually shape the lifestyle of the next century.

This dissertation examines two of the most fundamental problems in computer networks: ***routing*** and ***congestion control***. Routing can be defined as the process of selecting the best paths for the traffic flows in the network. Congestion control is concerned with regulating traffic flows to protect the network from traffic overloads. Routing and congestion control constitute a natural system of checks and balances for network traffic control. On one hand, routing attempts to find the best resources for the traffic. On the other hand, congestion control is designed to limit traffic load and avoid overloads. If the network is considered as a resource shared by competing users, routing and congestion control can also be viewed as two closely related processes of resource allocation. Operating on different geographic and time scales, routing and congestion control together form a resource control system which manages the entire network resource. While routing decides the grand strategy for overall network sharing, congestion control determines the sharing of resources at each bottleneck.

John McQuillan [McQu79] put this distinctions between routing and congestion control nicely:

> Routing is inherently a macroscopic process; that is, it should deal with the average trends in the traffic flow, since it is not appropriate to change the routing strategy used in a network to deal with momentary traffic fluctuations. For this reason, the routing procedure is best performed by some form of global algorithm which has information about the conditions throughout the entire network. Congestion control, on the other hand, is inherently a microscopic process, since it should deal with the variance in traffic flows, often in a small section of the network, in addition to the average or expected traffic flow. Congestion is caused when the computer processing power, memory storage, or line bandwidth available at a particular node is not adequate to deal with the traffic flow entering that node. Thus, congestion control is inherently local in nature, since it must deal with moment-to-moment fluctuations which may be visible only at the local level. This is not to say, of course, that a more global algorithm should not be constructed in addition to this local control.

There are two separate problems in this resource control system: (1) how to determine the optimal allocation for a given static condition, and (2) how to cope with dynamically changing conditions.

The first problem is often referred as optimal routing and flow control [Bert87] and is in fact a special case of the multicommodity network flow problem in operation research studies. Real optimal algorithms are often impractical for real networks due to their computational complexity and excessive information exchange. The algorithms in which we are interested in this dissertation are those algorithms that are based on simplified assumptions yet yield sufficiently good results for real networks.

In real networks, the conditions are changing constantly, as a result of link or node failures, setup or teardown of connections and natural fluctuations of traffic sources. It is therefore of vital importance for routing and congestion control to adapt to changing conditions. However, this problem of adapting to changing conditions is difficult, particularly in datagram networks where allocation and de-allocation of resources are more dynamic. The difficulty is primarily caused by the distributed nature of computation, the incomplete state information, the delay in information exchange and the unpredictable traffic characteristics. Much of the research in this dissertation is devoted to solving this problem of adapting to changing conditions.

In the rest of this chapter, we look at the development of computer networking and the research on routing and congestion control from a historical perspective. Technical details are

dealt with in the next two chapters.

1.1. Evolution of Packet Switching

1.1.1. Early History

There have been always two fundamentally different approaches in communication technology: pre-allocation and dynamic-allocation of transmission bandwidth. The telephone, telex and fax are circuit-switched systems, where an end-to-end circuit has to be set up before any data can be transmitted. Therefore a fixed bandwidth is preallocated for the duration of a call. On the other hand, telegraph and electronic mail have been operated by dynamically allocating bandwidths. When a message is sent, no attempt is made to schedule bandwidth over the entire source-to-destination path.

Although dynamic-allocation was much more efficient in using bandwidth resources than pre-allocation, almost all interactive data networks were dominated by circuit-switched systems before the 1970's. However, with the rapid advances in computer technology, dynamic-allocation techniques were made feasible and demonstrated substantial economic and performance advantages.

Packet switching was the result of this continuous evolution of dynamic-allocation techniques. In a packet-switched network, the flow of information is divided into segments, or packets, which are successively stored and forwarded through the network. Bandwidth is acquired and released at each switch when it is needed.

The development of packet switching has shown that it can not only improve the efficiency in bandwidth usage, but also enhance the reliability and flexibility of data communications. In the 1970's, packet switching became accepted worldwide for data communication networks.

1.1.2. The Pioneers

The first systematic study of packet switching was performed by the Rand Corporation in the early 1960's. An 11-volume analysis, *On Distributed Communications*, prepared by Paul Baran, was published in 1964 [Bara64]. This study proposed a fully distributed packet-switched network to provide for all military communications, data and voice. Extensive studies were carried out on the survivability of the network after heavy node and link failures. The primary goal was to maintain high performance in the face of hostile attacks on the network. Architecture and

implementation details were also investigated. Rand's study concluded that a large scale packet-switched network was not only feasible but also highly cost-effective. Unfortunately, Rand's report was largely ignored for several years until packet switching was rediscovered and applied by other people.

Also in the 1964 - 1965 period, Lawrence Roberts of the Advanced Research Projects Agency (ARPA) in the US and Donald Davies of the National Physical Laboratory (NPL) in the UK were considering to link time-sharing computer systems through communication systems [Robe78]. In autumn 1965, a seminar was sponsored by NPL to discuss the issues. Soon after that Donald Davies circulated his proposal of designing a packet-switched network throughout the UK. In June 1966, he used the term *packet* for the first time to describe the 128-byte blocks travelling inside the network. It was at that time that Donald Davies discovered the Baran's 1964 report. The NPL proposal was first published at the ACM Symposium in Gatlinburg, US, in October 1967. The proposal was similar to that in Baran's report and the analysis showed strong economic advantages of packet switching. Unfortunately, the communication industry was not convinced and for many years no large scale systems were developed. Nevertheless, Donald Davies was able to build a single packet switch at NPL. By 1973, this network was extended to connect 18 branches of NPL and was providing data distribution service for the laboratory.

Meanwhile, in the US a project to build the ARPANET to link computers at leading universities and research labs was planned under the sponsorship of ARPA [Robe78, McQu77]. In June 1967, the plan was first published at the ACM Symposium in Gatlinburg along with the NPL proposal. In January 1969, Bolt Beranek and Newman, Inc. (BBN) was awarded the contract by ARPA to develop the network switch *Interface Message Processor (IMP)* for the ARPANET. In September 1969, an one-node packet-switched network came to life at UCLA. Shortly after that four nodes were installed and operating effectively. The ARPANET grew rapidly. By 1977, it already had 111 hosts. As many universities and research establishments on the ARPANET later had their own local area networks and these were also connected to the ARPANET, the ARPANET eventually became the core network of the ARPA Internet, an internetwork of many networks running the TCP/IP protocol suite. In 1990, after more than twenty years' operation, the ARPANET (Net 10) was at last retired. However, its functions have already been replaced by various higher speed networks.

The ARPANET experiment is of great importance in the history of data networking. Much of our present knowledge about networking is a direct result of the ARPANET project. The

success of the ARPANET demonstrated that packet switching could provide efficient and reliable data communications. The research and implementation experience resulted in extensive publications which had significant influence on the design of subsequent computer networks. In particular, the work of Leonard Kleinrock and his fellow researchers at UCLA on the theory and measurement of the ARPANET provided a firm theoretic and practical understanding of the packet-switched networks.

In the 1970's, packet-switched networks gained worldwide acceptance and grew rapidly [Robe78]. In France, the interest in packet switching resulted in an experimental network CYCLADES being built in 1973 to link major computer centers in the country. The concept of *datagram* was introduced for the first time. The communication subnet, called CIGALE, considered each packet (called datagram) an independent unit and delivered it without any concept of messages, connections or flow control. The rest of the network functions were embedded in the hosts.

In the UK, universities and research establishments were connected through the Science and Engineering Research Council Network (SERCnet) in 1977, which was later renamed as Joint Academic NETwork (JANET). A set of protocols called *Coloured Book Protocols* were developed in 1979.

In the same period, many large corporations such as Xerox, IBM, DEC and AT&T began to build their internal networks for their business operations. Communities with similar interests set up their own cooperative networks like BITNET, UUCP and ACSNET [Quar90].

The development of private packet-switched networks encouraged some of the public carriers to build their own packet networks. In early 1970's, the British Post Office planned the Experiment Packet Switched Service (EPSS). In 1973, French PTT announced its plan to build TRANSPAC. The next year the Trans-Canada Telephone System announced DATAPAC. In US, Tymshare Corporation started in early 1970's to install a network for asynchronous timesharing terminals to access the central computers which was later developed into TYMNET. In late 1972, BBN formed Telenet Communication Corporation to provide public data network service. In 1976 the CCITT ratified the X.25 data network access protocols which public data networks universally adopted. Most public data networks are based on the *virtual circuit* approach which was developed by telecommunication community from their circuit switching bases.

1.1.3. Recent Trends in Packet Switching

There have been long debates over the two packet switching approaches: virtual-circuit and datagram. In a virtual-circuit network, a connection has to be set up before the data can be sent. During the set-up, a path across the network from the source to destination is specified and buffers at the intermediate nodes may have to be reserved. All packets belonging to the same connection contain the same circuit number and traverse the network via the established path. Flow control and error recovery are often performed on a hop-by-hop basis. The data is guaranteed to be delivered correctly in an orderly and flow controlled way. Since the packets contain circuit numbers instead of full source and destination addresses, the control overhead is low. In a virtual circuit network, state information is maintained at each intermediate node, which makes the network more vulnerable to network failures, but the flow control and accounting can be carried out more easily. In a datagram network, each packet, or datagram, has full source and destination addresses and is treated as an independent entity. Each datagram is routed to its destination individually therefore packets between the same source and destination may follow different paths and packets may not be guaranteed to arrive in the same sequence in which they left. Flow control and error recovery are usually handled on an end-to-end basis. In a datagram network, there is no need for connection set-up and no state information held inside the network therefore the network can survive node and link failures and adapt to dynamic traffic changes. However, traffic control in datagram networks is often more difficult, and it can be harder to quantify the Quality of Service.

Recent development in fast packet switching has made the difference between virtual-circuit and datagram much less distinct. Fast packet switching attempts to retain the merits of both virtual-circuit and datagram and provides an integrated network service to a large variety of applications.

Fast packet switching is a concept that covers several alternatives, all with the same basic characteristic, ie. packet switching with minimal functionality in the network. Optical fibre transmission technology can now provide very high capacity links with very low bit error rate. Therefore flow control and error recovery are no longer necessary inside the network and can be dealt with on an end-to-end basis. This substantially reduces the software complexity in the switches and enables more efficient switching. Furthermore, voice and video can tolerate a certain degree of errors. Fast packet switching operates in a connection-oriented mode so that resource reservation and performance guarantees can be implemented effectively to meet

stringent delay and jitter requirements of real-time applications. Fast packet switching systems often use short fixed packet size and limited header functionality. This allows the header processing to be done at a very high speed and to be implemented in hardware. Fast packet switching with short fixed length packets is often referred to as *asynchronous transfer mode (ATM)* in the context of broadband ISDN [Coud88].

1.2. Research On Routing and Congestion Control

1.2.1. Routing

The theory of routing in traditional circuit-switched telephone networks was well studied in 1950's. However, although some abstract concepts of circuit-switched networks are applicable to packet-switching networks, they are in general very different.

Another field which is closely related to routing in packet switching is graph theory. Many of the algorithms in graph theory can be directly applied to routing in packet switching. Nevertheless, routing problems in graph theory are usually static and many algorithms are too complex to be used in real networks.

From the very early beginning, routing in packet switching emphasized the importance of survivability in the presence of resource failures. In Baran's report, several adaptive routing schemes were proposed. One simple scheme is called *hot-potato*. When a packet arrives, the switch tries to get rid of it as fast as it can, by putting it on the shortest output queue. Baran also suggested the *backward learning* technique to gather routing information by examining the packets received.

In the ARPANET experiment, routing algorithms for packet switching were exploited extensively by various research groups [McQu74]. The Network Measurement Centre (NMC) under the direction of Professor Leonard Kleinrock conducted many influential studies on analysis and measurement of network performance. In his PhD thesis at MIT [Klei64], *Communication Nets: Stochastic Message Flow and Delay*, Kleinrock developed a precise model for networks and examined various aspects of network operation. In his thesis, random routing and fixed routing were investigated and compared. Another important study was carried out at NMC by Gary Fultz. In his PhD thesis, *Adaptive Routing Techniques for Message-Switching Computer-Communications Networks*, he developed a classification scheme and compared the performance of several routing algorithms, in particular, the effects on message delays

[McQu74]. Another research group at the Network Analysis Corporation, led by Dr. Howard Frank and Dr. Ivan Frisch, were responsible for the topological design of the ARPANET. Much of their work was closely related to the routing problem.

However, the first adaptive routing algorithm for use in a packet-switched computer network was designed and implemented by the BBN IMP group for the ARPANET project. The routing algorithm was fully discussed in the PhD thesis of one of the principal designers John McQuillian [McQu74] at Harvard University, *Adaptive Routing Algorithms for Distributed Computer Networks*. McQuillian's thesis also investigated various aspects of routing in detail and is a classic reference for routing.

The ARPANET routing algorithm is based on one attributed to Ford and Fulkerson [Ford62], which is often called the *distance-vector* algorithm. When applying Ford and Fulkerson's algorithm to an adaptive environment, there is a problem of long-lasting routing loops in the face of link distance increase. In the ARPANET algorithm, a simple technique called *hold-down* was used to eliminate some of the looping problems [McQu74].

Due to the problems of routing loops and delay measurement, BBN later developed a new routing algorithm called *Shortest-Path-First (SPF)* [McQu80] and replaced the original ARPANET routing algorithm in May 1979. The new ARPANET routing algorithm is a decentralized implementation of the Dijkstra's algorithm [Dijk59]. It is often called the *link-state* algorithm. The new ARPANET routing algorithm eliminated many of the problems associated with the old ARPANET algorithm and greatly improved the routing performance.

The old and new ARPANET routing algorithms are of great importance in routing algorithms for packet-switched networks. They present two basic classes of shortest-path routing algorithms which are widely used in today's computer networks. We will discuss shortest-path routing algorithms in more detail in chapter three.

In the 1980's, a considerable amount of efforts were made on distance-vector algorithms to eliminate the problem of routing loops. One of the techniques called *split-horizon* proposed by Cegrell [Cegr75] was exploited by Stern, Shin and Chen, and Garcia-Luna-Aceves [Schw80, Shin87, Garc86]. However, none of the routing algorithms using the spilt-horizon solved the problem of routing loops completely. Merlin and Segall [Merl79] introduced the concept of *coordinated update* to eliminate the looping problem. Jaffe and Moss [Jaff82] later proposed a more responsive algorithm based on the same concept. Recently Garcia-Luna-Aceves published several algorithms which further developed this idea [Garc88, Garc89a, Garc89b].

Garcia-Luna-Aceves considered the distributed routing loops as a problem of *diffusing computations* and used a technique developed by Dijkstra and Scholten [Dijk80] and produced a distance-vector algorithm which is loop-free after an arbitrary sequence of topological changes.

Flooding and source routing are two special forms of routing that are also widely used [Bert87]. In flooding, each packet is sent on every outgoing line except the line it arrived on. Some mechanisms such as sequence number are often used to detect duplication and terminate the propagation. Flooding does not require any information about the topology and is extremely robust therefore it is often used for distributing routing and other control information. In source routing, the source makes the routing decisions and specifies the route in the packets that it sends. To determine the route, the source must have information about the topology. Source routing is very useful for some control and debugging purposes.

More recently a new form of routing called *Policy Based Routing (PBR)* [Clar90] has been proposed. In an internetwork composed of separately administered domains, routing must be sensitive to the human concerns and administrative inputs. PBR is used as an integral component for managing resource and access control in such an environment.

The algorithms discussed so far are largely empirical. In [Bert87], Bertsekas and Gallager presented some algorithms that are provably optimal under certain limited conditions. In some networks, the data flows are relatively stable and predictable. Under the condition that the traffic patterns are known in advance and approximately constant in time, it is possible to optimally distribute the traffic so that the mean delay for the whole network is minimized.

1.2.2. Congestion Control

It was recognized from the very beginning that unrestricted flows of packets may overload the network and lead to congestion. Network congestion, if not prevented, can progressively degrade the performance of the network and ultimately lead to a deadlock.

In early ARPANET, a centralized algorithm called *flow deviation method* [Gerl80] was used to regulate the flows. The flow deviation method assigns flows within a network to minimize cost, delay or both based on topology information and external flow requirements reported by each node in the network.

At the early days, memory was the scarce resource and networks were built with the minimum amount of memory required for operation. There were many buffer management schemes proposed to prevent buffer deadlock and achieve optimal buffer allocation [Gerl80].

Control mechanisms were implemented at the entry points of the network to prevent excessive numbers of packets from being stored in the network. One scheme, which is based on the concept of *permits*, is called *Isarithmic Scheme* [Pric77]. Under the isarithmic scheme, the network is initially provided with a number of permits, several held in each node. Each packet has to obtain a permit at the source and release the permit when it arrives at the destination. The isarithmic scheme works well in the network where traffic patterns are stable and predictable.

Another scheme called *Choke Packet* is a feedback control algorithm [Maji79]. When congestion occurs on an intermediate node, a choke packet is sent back to the source instructing it to block further transmission. The source can resume the transmission when no choke packets are received during a specified time interval. A similar feedback mechanism called *Source Quench* is also provided in the Internet Protocol [Prue87].

Most of the transport protocols used window-based mechanisms for controlling end-to-end buffer congestion [Gerl80]. With a window-based scheme, the source has to obtain permission from the destination before sending a new group of packets. The number of packets that the source are permitted to send is called *window size* or *credits*. The destination may decide the window size based on local buffer availability. Besides presenting destination buffer congestion, window-based mechanisms also indirectly provides some degree of congestion control for the intermediate nodes, since congestion in the network often causes longer end-to-end delay, thus slows down the return of credits to the source. In some networks, intermediate nodes can modify the window size explicitly. Many transport protocols combine congestion control, error recovery and retransmissions into one window-based mechanism [Gerl80].

At the time when the use of networks was very much limited and the traffic patterns were stable, most of these schemes proved to be adequately effective. However, as the networks grew in both use and size, congestion control became a serious problem, particularly in datagram networks where resources are not usually reserved in advance.

In 1984, Jain, Ramakrishnan and Chiu at DEC started a project on congestion control and developed three feedback control schemes [Rama88, Jain89, Jain86]. All three schemes used the same traffic adjustment algorithm called *additive increase and multiplicative decrease (AIMD)* [Chiu89]. The idea is that each user gains a equal share in the increase operation and lose in proportion to the share it has in the multiplicative decrease operation. Therefore the users who have larger than fair shares lose some advantage in each iteration and eventually all users oscillate in a range near the optimal point. The feedback mechanisms used by the schemes to determine the

increase or decrease operation are different and based on timeout, round trip delay and explicit congestion bit in the packet header.

In October 1986, the Internet experienced a series of congestion collapses. The effective throughput of some links dropped significantly. As a result of an investigation, Jacobson added several new algorithms to TCP [Jaco88]. One algorithm called *slow-start* is used to avoid overwhelming a slow bottleneck with bursts of back-to-back packets when the connection starts or restarts after a timeout. The congestion avoidance algorithm added to TCP is in fact a timeout-based additive increase and multiplicative decrease algorithm.

It has come to be recognized that congestion control is a problem of resource allocation. As there is a limit on the lifetime of a packet, increasing buffer size does not solve the problem of congestion. The network has to actively control the traffic to enforce fair resource sharing and to protect well-behaved users [Nagl87].

In 1985, Nagle proposed a network control algorithm called *Fair Queueing*, which was later improved and extensively studied by Demers, Keshav and Shenker [Nagl87, Deme89]. In the fair queueing algorithm, each node maintains separate queues for each conversations and serves them in a round-robin manner. Fair queueing algorithm ensures the bandwidth is allocated fairly among different conversations.

Zhang proposed a control algorithm called *Virtual Clock* which aims to emulate Time Division Multiplexing (TDM) [Zhan90]. Virtual Clock prioritises packets from different flows according to their virtual clocks based on TDM. It has been shown that Fair Queueing and Virtual Clock are in fact equivalent [Zhan91a].

Another approach proposed is called *Leaky Bucket* [Turn86]. In Leaky Bucket, each connection is associated with a counter which is incremented whenever the user sends a packet and is decremented periodically. If the counter exceeds a threshold, the network discards the packet. The rate at which the counter is decremented and the value of the threshold determine the average bandwidth and burstiness respectively.

1.3. Organization of Dissertation

The rest of the dissertation is organized as follows:

Chapter two introduces network architectures. It first outlines the principles of layering and the ISO OSI Reference Model then discusses the DARPA Internet model and its protocol suite.

Chapter three presents a discussion on the characteristics of routing and congestion control. Two shortest path routing algorithms and two end system congestion control schemes which are widely used in real networks are analyzed.

Chapter four looks at the behavior of shortest path routing algorithms in a dynamic environment. The shortest path routing algorithms are examined from the perspective of control theory and decision making and six major problems are identified and analyzed in detail.

Chapter five introduces a new approach for dynamic routing based on the concept of decentralized decision making. A new dynamic routing algorithm based on this approach is described. Results of extensive simulation are presented.

Chapter six examines the information feedback mechanisms in congestion control. Two new binary feedback schemes and one new quantitative feedback scheme are proposed.

Chapter seven proposes a new traffic adjustment algorithm. Three new end-system congestion control schemes, based on the new traffic adjustment algorithm, are described. Performance comparison based on extensive simulation is presented.

Chapter eight summarizes the contributions of this research and outlines the future work.

CHAPTER TWO

COMPUTER NETWORK ARCHITECTURE

This chapter introduces network architectures. It first outlines the principles of layering and the ISO OSI Reference Model then discusses the DARPA Internet model and its protocol suite.

2.1. Computer Network Architecture

2.1.1. Network Layering

To reduce the design complexity, computer networks are usually organized in a highly structured way. Most networks are designed as a series of *layers* or *levels*, each built upon its predecessor and performing specific functions. This modular approach allows the constituent activities to be more easily visualized and individual parts to be independently developed.

Computer networks are complex communication systems which involve more than one computer systems. There must be a set of common rules for generating and interpreting the messages they send and receive. A structured approach to defining these rules is needed to manage the complexity of specification and design.

In network architectures, the network protocols are often divided into a series of layers of function. A *layer* is created where a different level of abstraction is needed. Each layer performs a set of well-defined functions. The term *service* is often used to represent the definition of such a demarcation and boundary between the functions. By convention, the layers are viewed as a vertical sequence, with the most abstract statements of objectives at the top and successively more specific, technology-dependent functions. Each layer adds to the functions of the lower layer service in order to provide service to higher layer.

The service provided to the layers above, a statement of the set of offered capacities, is independent of the detailed way in which they are realized. The definition of a service only

establishes the properties of a type of communication and the relationship between the user of the service and the service provider. A service can be implemented in many different ways. However, it should exhibit the same behavior to the outside world as that is described by the definition.

The choice of a division into a specific set of layered services is not unique. It is an engineering decision that has to be made based on a balance between many conflicting factors.

The International Standards Organization (ISO) Open Systems Interconnection (OSI) Reference Model [Folt83] is an example of this layering technique. In the OSI Model, ten guiding principles of layering were drawn-up, which are summarised below:

a) keep the overall structure simple;

b) create boundaries only at points where interactions across the boundaries are minimized;

c) separate into different layers those that are very different in nature and purpose;

d) collect together within a layer those functions that are similar or highly related;

e) select boundaries at points which past experience has shown to be successful;

f) create layers such that the internal mechanisation of such layers can be managed independently from the functionality that it provides;

g) create boundaries where it may be useful to have a real physical boundary;

h) create a layer where there is a need for a different level of abstraction in handling of data;

j) create layers such that changes of functions and protocols can be made without affecting other layers;

k) create for each layer boundaries with its upper and lower layer only.

It is often necessary to further subdivide individual layers into sublayers and the basic principles described above can be also applied. In addition, three more principles are given for sublayering:

m) create further subgrouping and organization of functions to form sublayers in cases where communications services need it;

n) create, where needed, two or more sublayers with a common and therefore minimal functionality to allow interface operation with adjacent layers;

p) allow by-passing of sublayers.

We discuss below some basic considerations of network layering that has led to the OSI architecture.

The basic function of the network is to provide a reliable transportation of data between end-users and to support meaningful communication between them. The end-users are often application programs such as mail-systems, airline seat reservation systems and information service systems. We usually refer them as applications. Applications will be many and various. The architecture must not only accommodate existing applications but also those will be introduced in the future.

One important component in the network is the telecommunication medium, e.g. leased line, public switched network, satellite communication network, etc. Those which are often referred as physical networks are various and evolving rapidly. It is clearly not acceptable that applications have to be redesigned for each new physical networks. It is also desirable that networks based on different physical network technologies can be interconnected together.

The above consideration led to the fundamental division in the architecture to provide a function boundary over which any application can operate irrespective of any underlying physical network technologies. This division occurs at the Network Layer of the OSI Model. The Network Layer masks from the upper layers any peculiarities of the actual physical networks. The Network Layer also must be able to interconnect networks of different physical network technologies in such a way as to provide services to upper layers like a single network.

In addition to the service that Network Layer provides, the network also has to provide appropriate Quality of Service (QoS) to meet the different requirements of the applications such as priorities, reliability and delay bounds. This leads to the Transport Layer which provides the QoS required by the applications in a cost-optimized fashion.

The architecture below the Network Layer typically reflects the architectures in public switched networks. It contains two layers: the Physical Layer for physical and electrical characteristics of the interface to the network, and the Data Link Layer for the data transmission procedures to transfer information to the switch.

Above the Transport Layer, there is a Session Layer which establishes the relation between the two end-users, maintains the integrity of it and controls the data exchange between the two partners. As different computers have different representation of data, the data structure to be exchanged is often defined in an abstract way. It is the Presentation Layer that is responsible for managing these abstract data structures and converting from the representation inside the

computers to the network standard one.

2.1.2. OSI Reference Model

The OSI Reference Model was developed as a basic framework to deal with the complexity of communicating systems and the required protocol standards. In this section, we first examine the basic concepts in the architecture and then discuss the Seven-Layer Model in detail.

2.1.2.1. OSI Terminology

Systems, Layer and Entities

The OSI Model is an abstract description of interprocess communications. It is concerned with standards for communications between systems. In the OSI Model, communications take place between application processes running in different systems. A *system* is viewed to be one or more autonomous computers and their associated software and users that are capable of information processing and transfer. The internal operation of a system is not of interest to the OSI Model.

Layering is used as a structuring technique to allow the open systems to be logically decomposed into independent, small subsystems. Each individual system itself is viewed as being logically composed of a succession of subsystems. Each subsystem, in turn, is viewed as being made of one or several *entities*. A *layer* comprises many entities distributed among interconnected open systems. Entities in the same layer are called peer entities.

For simplicity, any layer is referred to as the (N)-layer, while its next lower layer and next high layer are referred to as the (N-1)-layer and the (N+1)-layer respectively. Similarly, the entities in the (N)-layer are termed (N)-layer entities.

Services and Service Access Points

A *service* is a capability of the (N)-layer which provided to the (N+1)-entities. It is important to note that only those capabilities that can be seen from the layer above are services.

The (N)-entities add value to the (N-1)-service that they get from the (N-1)-layer then offer the (N)-service to the (N+1)-entities. The point that the (N)-services are offered to the (N+1)-entities is called (N)-service access point, or (N)-SAP for short. The (N)-SAPs represents the logical interfaces between the (N)-entities and the (N+1)-entities. An (N)-SAP is located by its (N)-address and can be served by only one (N)-entity and used by only one (N+1)-entity.

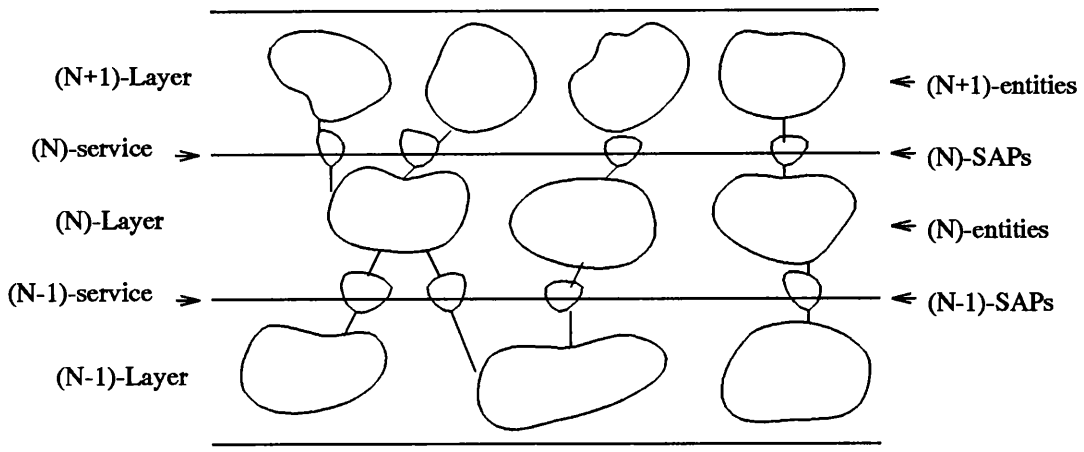


Figure 2.1: The Service Model

Modes of Operation

There are two different kinds of service: the connection-orientated mode (COM) and the connectionless mode (CLM).

The connection-orientated model is based on classical sign-on and sign-off mode of operation. For this type of operation the associated (N)-Layer would clearly have a connection-establishment phase, a data transfer phase and a disconnection phase. The association is established for the duration of the data transfer and is created at establishment time and destroyed at disconnection time.

Connectionless mode, on the other hand, does not have three phases of operation. There is no connection establishment or disconnection phase, and no negotiation of the association between communicating phase. Each data unit exchanged is totally self-contained, ie. it has all the necessary information associated with it.

2.1.2.2. The Seven-Layer Model

In OSI Model, the network architecture is divided into seven independent layers. These layers are illustrated in Figure 2.2 The highest is the Application Layer which consists of the application-entities that co-operate in the OSI environment. The lower layers provide a step-by-step enhancement of communication services. The boundary between two layers identifies a stage in this enhancement of services at which an OSI service standard is defined.

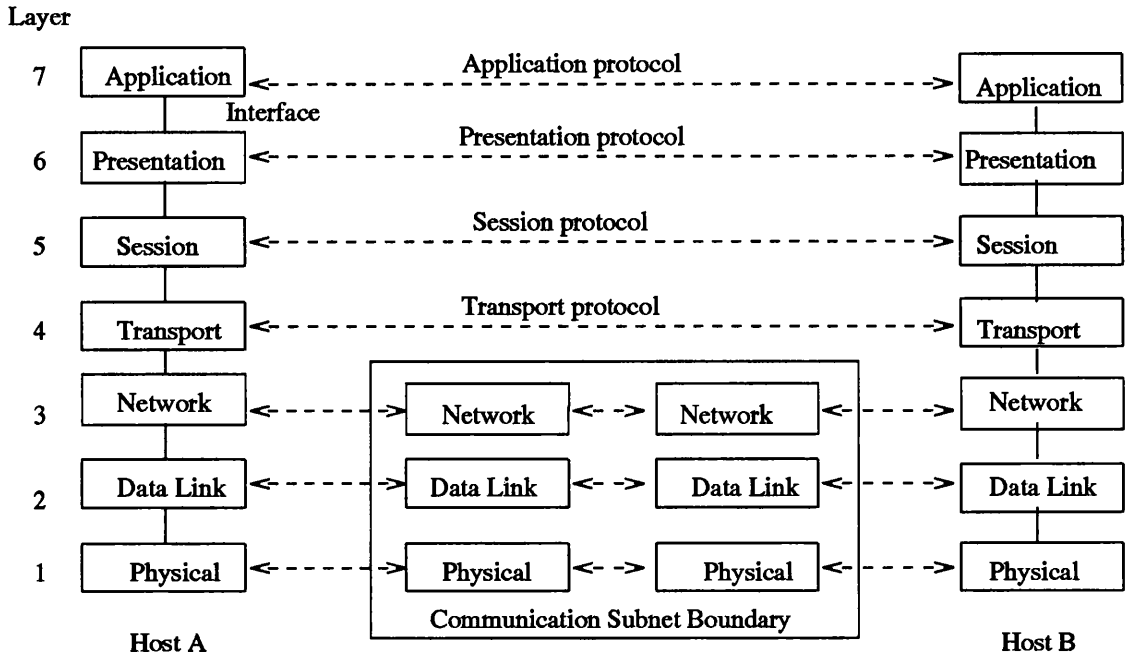


Figure 2.2: The Seven-Layer Model

Physical Layer

The Physical Layer provides the mechanical, electrical, functional and procedural means to activate, maintain and deactivate physical connections for bit transmission between data-link-entities. A Physical connection may involve intermediate open systems, each relaying bit transmission within the Physical Layer. Physical Layer entities are interconnected by means of a physical medium.

The following services are provided the Physical Layer:

a) physical-connections;

b) physical-service-data-units;

c) physical-connection-endpoints;

d) data-circuit identification;

e) sequencing;

f) fault condition notification; and

g) quality of service parameters.

The following functions are performed within the Physical Layer:

a) Physical connection activation and deactivation;

b) Physical Service data unit transmission; and

c) Physical layer management.

Data Link Layer

The purpose of the Data Link Layer is to provide the functional and procedural means to establish, maintain and release data link connections among network-entities and to transfer data-link-service-data-units. The Data Link Layer also detects and possibly corrects errors which may occur in the Physical Layer. The services provided by the Data Link Layer are described as follows:

a) data-link-connection;

b) data-link-service-data-units;

c) data-link-connection-endpoint-identifiers;

d) sequencing;

e) error notification;

f) flow control; and

g) quality of service parameters.

The set of required functions within the Data Link Layer are described below:

a) data-link-connection establishment and release;

b) data-link-service-data-unit mapping;

c) data-link-connection splitting;

d) delimiting and synchronization;

e) sequencing control;

f) error detection;

g) error recovery;

h) flow control;

j) identification and parameter exchange;

k) control of data-circuit interconnection; and

m) Data Link Layer management.

Network Layer

The Network Layer, sometimes called the communication subnet layer, provides independence from the data transfer technology and independence from relaying and routing considerations. The Network Layer masks from the Transport layer all the characteristics of the actual transfer medium.

The basic service of the Network Layer is to provide the transparent transfer of the data between transport-entities. The quality of service is negotiated between the transport-entities and the network-service at the time of establishment of a network-connection.

The following services provided by the Network Layer are described below:

a) network-address;

b) network-connections;

c) network-connection-endpoint-identifiers;

d) network-service-data-unit transfer;

e) quality of service parameters;

f) error notification;

g) sequencing;

h) flow control;

j) expedited network-data-unit transfer;

k) reset; and

m) release.

The Network Layer functions can be categorized into three sublayers:

3C: the Concatenation and Routing Functions, which are concerned with routing and relaying across concatenated networks.

3B: the Subnetwork Convergence Functions, concerned with the functions necessary to enhance a particular subnetwork to allow data transfer across it to meet the requested quality of the service parameters.

3A: the Subnetwork Access functions concerned with directly using the data link service to provide an abstract subnetwork.

The following are functions performed by the Network Layer:

a) routing and relaying;

b) network-connections;

c) network-connection multiplexing;

d) segmenting and blocking;

e) error detection;

f) error recovery;

g) sequencing;

h) flow control;

j) expedited data transfer;

k) reset;

m) service selection; and

n) network layer management.

Transport Layer

The purpose of the Transport Layer is to provide transparent transfer of data between end systems, thus relieving the upper layers from any concern with providing reliable and cost effective data transfer. The Transport Layer optimizes the use of the available network services to provide the performance required by the Session Layer at minimum cost. All protocols defined in the Transport Layer have end-to-end significance.

The Transport Layer provides both the connection-mode service and the connectionless-mode service. The essential features of the connection-mode service are:

a) transport-connection establishment;

b) data transfer; and

c) transport-connection release.

The connectionless-mode has only one essential service: data transfer. The connectionless-mode operation provides for the simple transfer of data, together with control data and QoS parameters so that only a single operation is needed.

The Transport Layer functions include:

a) mapping transport-address onto network-address;

b) multiplexing transport-connections onto network-connections;

c) establishment and release of transport-connections;

d) end-to-end sequence control on individual connections;

e) end-to-end error detection and recovery;

f) end-to-end quality of service monitoring

g) end-to-end segmenting, blocking and concatenation;

h) end-to-end flow control on individual connections;

j) supervisory functions; and

k) expedited transport-service-data-unit transfer.

Session Layer

The primary purpose of the Session Layer is to provide the mechanisms for organizing and structuring the interactions between application processes. The mechanisms provided in the Session Layer allow for two way simultaneous and two way alternate operation, the establishment of the major and minor synchronization points, and the definition of special tokens for structuring exchanges. In essence, the Session Layer provides the structure for controlling the communication.

The services provided by the Session Layer include:

a) session-connection establishment;

b) session-connection release;

c) normal data exchange;

d) quarantine service;

e) expedited data exchange;

f) interaction management;

g) session-connection synchronization; and

h) exception reporting.

The functions within the Session Layer to provide the service are given below:

a) session-connection to transport-connection mapping;

b) session-connection flow control;

c) expedited data transfer;

d) session-connection recovery;

e) Session Layer management.

Presentation Layer

The Presentation Layer provides for the representation of the information that application-entities either communicate or refer to in their communication. The Presentation layer is concerned only with the syntax and provides for a common representation to be used between application-entities. This relieves application-entities of any concern with the problem of representation of information, ie. it provides them with syntax independence.

The Presentation Layer provides the following services:

a) transformation of syntax; and

b) selection of syntax.

The Presentation Layer performs the following functions:

a) session establishment request;

b) data transfer;

c) negotiation and renegotiation of syntax;

d) transformation of syntax including data transformation, formatting and special purpose transformations; and

e) session termination request.

Application Layer

The Application Layer as the highest layer of OSI does not provide services to any other layer. It provides a means for the application processes to access the OSI environment. The purpose of the Application Layer is to serve as the window between correspondent application processes which are using the OSI to exchange meaningful information. All specifiable application process parameters of each OSI environment communications instance are made known to the OSI environment via the Application Layer.

The application-entities contains one users-element and a set of application-service-elements. The user-element represents that part of the application-process which uses those application-service-elements needed to accomplish the communications objectives of that

application-process.

The only means by which user-elements in different systems may communicate is through the exchange of application-protocol-data-units.

In addition to information transfer, the Application Layer also provides the following services:

a) identification of intended communications partners;

b) determination of the current availability of the intended communication partners;

c) establishment of authority to communicate;

d) agreement on privacy mechanisms;

e) authentication of intended communication partners;

f) determination of cost allocation methodology;

g) determination of the adequacy of resources;

h) determination of the acceptable quality of service;

j) synchronization of cooperating applications;

k) selection of the dialogue discipline;

m) agreement on responsibility for error recovery;

n) agreement on the procedures for control of data integrity; and

p) identification of constraints on data syntax.

The application-entities are organized into groups of functions. An application-process may determine the grouping of functions comprising the application-entity.

Two categories of application-service-elements are recognized: common-application-service-elements and specific-application-service-application-elements. The common-application-service-elements provide capacities that are generally useful to a variety of applications while the specific-application-service-application-elements provide capacities required by specific applications.

2.2. DARPA Internet Architecture

Before the OSI Model, there have been a number of network architectures based on real network implementations. Among them the DARPA Internet Architecture (DIA) [Come90], IBM's Systems Network Architecture (SNA) [Corr79], DEC's Digital Network Architecture (DNA)

[Weck80] and the Xerox Network System (XNS) [Xero85] have profound influence on the development of network architectures. In this section we focus on the DARPA Internet Architecture and its protocol suite as most of our simulation work on routing and congestion control in this dissertation is based on the Internet protocol suite.

2.2.1. Architecture Overview

Beginning with the ARPANET, DARPA has sponsored the development of a number of packet-switched networks including ARPANET, packet radio networks and satellite networks. All those networks running the TCP/IP protocol suite are collectively called the DARPA Internet.

The DARPA Internet protocol suite is designed to support communication between heterogeneous hosts and networks. A number of packet-switched networks are interconnected with gateways and each of the networks is assumed to be designed separately based on different technologies and requirements. It is assumed that each network is capable of transporting packets but delivery is not guaranteed. The end-to-end reliability has to be ensured by the end systems.

The DARPA Internet Architecture is different from the OSI Reference Model in two ways. First, the service provided by the network layer in the DIA is a connectionless datagram delivery. Reliability and connection management are all provided by the trahsport layer on an end-to-end basis. Second, there are no separate session layer and presentation layer in the DIA. The session and presentation layer functions are integrated into the applications.

The DARPA Internet Architecture only has four levels: Network Level, Internet Level, Service Level and Applications Level.

The Network Protocol level is the lowest level consisting of the physical, link, and network protocols. It provides the means for a host accessing the network. Therefore, the primary areas of concern to the Internet are the interface to the network and the performance offered by the network.

The next level is the Internet Protocol level which provides the mechanisms for connecting various networks and gateways into an internet system. The Internet Protocol (IP) is a datagram service, which unifies the available services into a uniform Internet datagram service. The IP includes functions such as global addressing, routing, fragmentation, assembly etc and it allows hosts to send packets through the Internet system without regard to the network on which the destination resides. In the Internet, the gateways take the responsibility for determining how to deliver the packets to other networks.

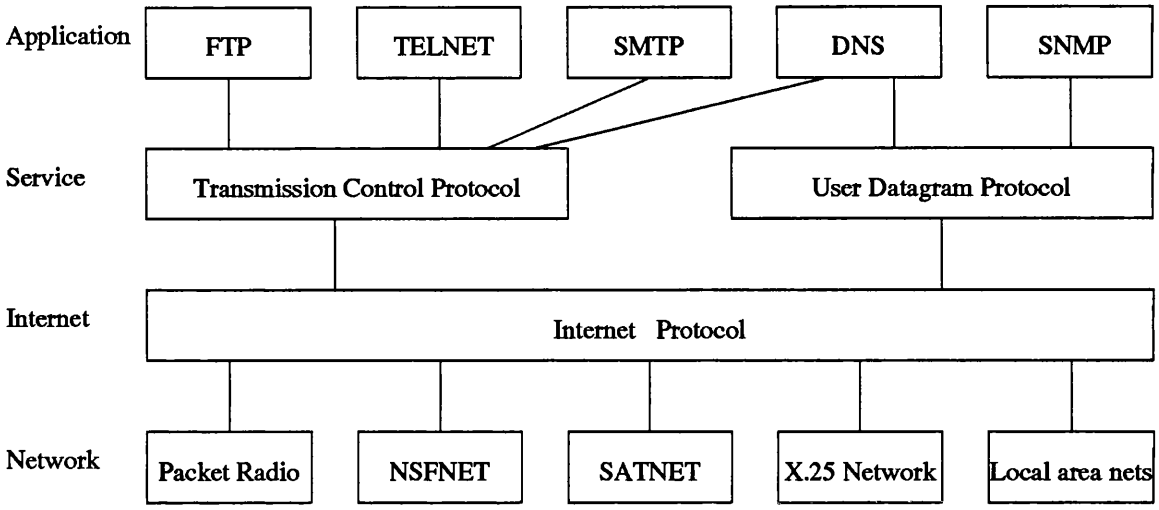


Figure 2.3: The DARPA Internet Model

The Internet Protocol provides an end-to-end datagram delivery service. The customer application, however, often requires a specific level of service. There are two major service level protocols in the Internet system: the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). In addition to end-to-end service protocols, there is a requirement for control in the Internet. The Internet Control Message Protocol (ICMP) has been developed for this purpose.

The TCP is an end-to-end reliable data stream protocol. It contains mechanisms including sequencing, checksumming, flow control, acknowledgements and retransmission procedures. The TCP provides a reliable, sequenced delivery of data to the applications.

Many applications do not require a reliable stream service. The basic datagram service of the Internet is sufficient for some applications if enhanced by services such as multiplexing and checksumming. The UDP provides such a service with peer-to-peer addressing and optional data checksums.

In such a complicated Internet, it is important to have monitoring and control capabilities. The ICMP provides these facilities to carry out control activity. it includes functions such as redirection, echoing, timestamping and indications of address masks and delivery problems such as destination unreachable and unreasonable parameter problems.

There are a number of Application Level Protocols that are provided by many TCP/IP implementations. Those include TELNET, FTP, SMTP, DNS and SNMP.

TELNET is a remote terminal access protocol. It allows an interactive user on a client system to start a login session on a remote system. Once a login session is established, the client process passes the user's keystrokes to the sever process. TELNET is based on TCP.

The File Transfer Protocol (FTP) is used to transfer files from one system to another. It provides user authentication, data conversion, binary and text modes, local and remote directory listings etc. For each session, FTP establishes two TCP connections between the client and server process - one for control information and the other for the data being transferred.

The Simple Mail Transfer Protocol (SMTP) is a protocol for exchanging electronic mail using a TCP connection between two systems. The Domain Name System (DNS) provides distributed hostname to IP address mapping service and some directory services. The Simple Network Management Protocol (SNMP) is designed for managing network components.

2.2.2. Internet Protocol (IP)

IP provides a connectionless, best-effort and unreliable delivery system [Post81a]. It is connectionless because it considers each IP datagram an independent entity. Each IP datagram contains the source and destination addresses so that each datagram is routed to its destination independently. The service that IP provides is unreliable because the delivery is not guaranteed. The packets may be lost, duplicated, or delivered out of order. Finally, it is called best-effort because IP makes an earnest attempt to deliver the packets. Since there is no state information stored inside the network, IP can make the best use of the redundant paths in the face of network failures.

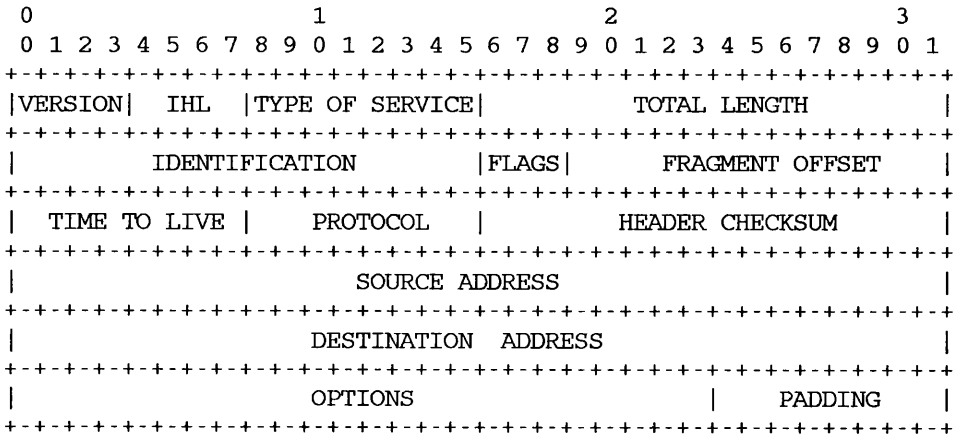


Figure 2.4: IP Header Format

The functions performed by IP are illustrated by the contents of its packet header, shown in Figure 2.4. The header identifies source and destination hosts and the destination protocol, and contains header and packet lengths. The identification and fragment field are used when a packet has to be broken into smaller sections for transmission on its next hop and to reassemble when they arrive at the destination. The Type of Service (ToS) field specifies how datagram should be handled. Although ToS is currently ignored in most implementations, this field provides a mechanism that will eventually allow some control over the quality of the service offered to different applications. The Time to live (TTL) field specifies how long, in seconds, the datagram is allowed to remain in the Internet. The datagram is discarded when its TTL reaches zero so that no datagram is allowed to travel around forever even if a routing loop forms. The Options field following the destination address is not required in every datagram. Most of the Options are used for network testing and debugging. Examples of existing options include record route, timestamp and source routing. Record route allows the source to create an empty list of IP addresses and arrange for each gateway that handles the datagram to add its address to the list. Timestamp is similar to the record route. However the entries in the timestamp field are the time and date at which a gateway handles the datagram. The source route option allows the sender to specify the path through the Internet. There are two forms of source routing. One form, called strict source routing, includes a sequence of Internet addresses that the datagram have to follow to reach the destination. An error results if a gateway can not forward the datagram to the next address on the list. The other form, called loose source routing, also includes a list of Internet addresses but multiple network hops are allowed between successive addresses on the list.

2.2.3. Transmission Control Protocol (TCP)

TCP provides a connection-oriented, reliable, full-duplex, byte-stream delivery service to an application program [Post81b]. A TCP connection can be viewed as a bidirectional, sequenced stream of data octets transferred between two peers. The stream delivery service on the destination host passes to the application the same sequence of octets that the sender passed to it on the source host. The stream initiation and termination are explicit at the start and end of the stream and they are also acknowledged in the same manner as data are. Because the packets may be lost, duplicated and delivered out of order, TCP has the task of recreating at the destination an exact copy of the data stream generated at the source, in the same order and with no gaps or duplicates. The mechanism used to accomplish this task is to assign a unique sequence number to each byte of data at its source, and to sort the bytes at the destination according to the sequence

number. The sorting operation corrects any disordering. An acknowledgement, timeout, and retransmission scheme corrects for data loss. TCP uses the sliding window mechanism to solve two important problems: pipelining and flow control. A TCP window mechanism allows multiple packets to be transmitted before any acknowledgement is received. The number of bytes that can be unacknowledged is called window size. By advertising the window size, the receiver can also restrict the transmission of packets when there are not enough buffer space at the receiving end. The acknowledgements are accumulative, ie. only the contiguous data received is acknowledged. When the retransmission timer expires, all unacknowledged data is to be resent.

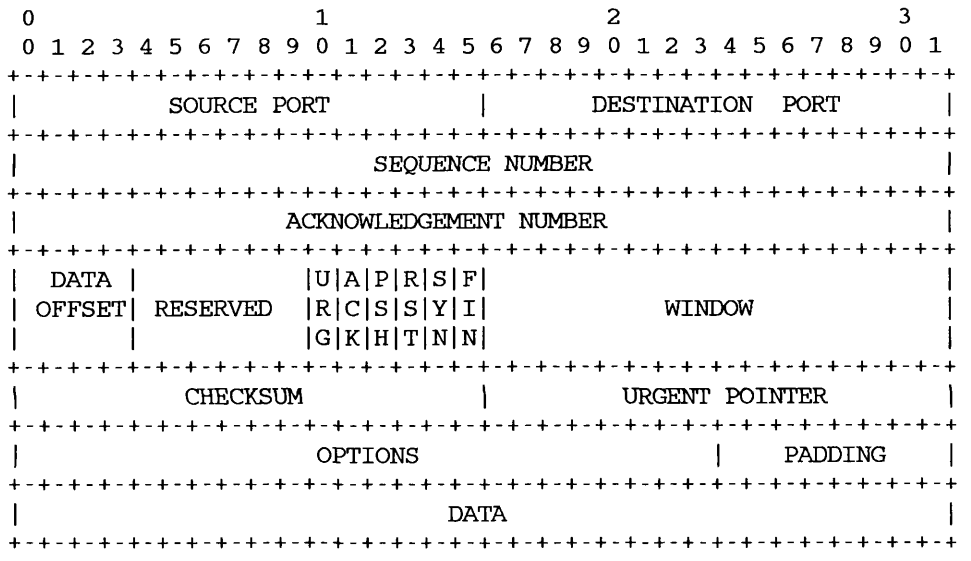


Figure 2.5: TCP Header Format

Figure 2.5 shows the TCP segment format. Each segment is divided into two parts, a header followed by data. The Source Port and Destination Port fields contain the TCP port numbers that identify the application programs at the ends of the connection. The Sequence Number identifies the position of the first byte in the data stream. The Acknowledgement Number field refers to the next byte not yet received in the opposite data stream. The Options field can be used to negotiate some parameters between the two ends of the TCP connection. Because Options field varies in length, the Offset field is used to indicate the offset of the data portion of the segment. The flag field is used to determine the purpose and contents of the segment. TCP allows the sender to specify that some data is *urgent* and should be delivered to the application as soon as possible. The Urgent Pointer field points to the position in the data stream where urgent data ends. TCP advertises how much data it is willing to accept by specifying its buffer size in the Window field.

2.3. Summary

The ISO OSI Reference Model provides a conceptual model for studying and describing network architectures. An overview of the Seven-Layer Model and the services offered by each layer are described. The major principles of the layering techniques are discussed.

The DARPA Internet Architecture has had profound influence on the development of computer networking. A brief introduction of the Internet protocol suite is presented. The two most important protocols IP and TCP are discussed.

CHAPTER THREE

ROUTING AND CONGESTION CONTROL

This chapter presents a discussion on the characteristics of routing and congestion control. Two shortest path routing algorithms and two end system congestion control schemes which are widely used in real networks are analyzed.

3.1. Routing

Routing is a network layer function that determines the paths from source to destination for traffic flows. The times at which routing decisions are made depend on whether the network uses datagrams or virtual circuits. In a datagram network, a routing decision is made for every incoming packet and the route to a destination can be changed at any time. In a virtual circuit network, routing decisions are made when a new virtual circuit is being set up. All data packets subsequently follow the established route until the session is terminated or reset.

The functions of a routing algorithm are of two levels. At the basic level, a routing algorithm has to maintain the reachability of the network. When parts of the network fails, a routing algorithm has to find alternative paths when they do exist. At a higher level, a routing algorithm has to ensure optimal and fair sharing of the network so that resources are efficiently utilized. The difficulty in routing is due to the distributed nature of the operation. A routing algorithm has to deal with resource failures and traffic changes with incomplete and delayed information feedback.

In this section, we first discuss the design goals, composition and classification of routing algorithms, and then examine two classes of shortest path routing algorithms in detail.

3.1.1. Design Goals

3.1.1.1. Efficiency

The operation of routing consumes network resources such as switch CPU, switch memory and link bandwidth. Routing control packets may also have higher priority in processing and transmission. It is important that routing algorithms are simple and efficient so that the processing and transmission of normal data packets are not affected. The efficiency can be measured in terms of computational complexity, storage complexity and communication complexity. In some cases precise measurement is difficult to obtain and the worst-case performance may be used as an indicator. A tradeoff has to be made between the functionality and the overhead.

3.1.1.2. Reliability

The routing algorithm is one of the critical components in the network. Its reliability and robustness are of vital importance. The behavior of a routing algorithm must be predictable. Malfunction in routing algorithms may have global implications and lead to catastrophic results. An example is the incident occurred in 1980 when the whole ARPANET collapsed due to what was diagnosed as a hardware malfunction which caused incorrect routing updates [Rose81]. It is desirable that a routing algorithm have the ability to carry out consistency checks and eliminate suspicious routing updates, so that it may survive both malfunctions and deliberate attacks.

3.1.1.3. Stability

Since routing is a distributed operation, stability is important. Inconsistent routing information or poor route computation can cause routing loops and generate large amounts of artificial traffic, which in some cases can bring down the network. For a given topology and traffic conditions the routing algorithm should eventually converge to a steady state free of routing loops and route oscillation. A routing algorithm has to be stable under arbitrary topologies and traffic conditions, in particular, under heavy and unevenly distributed traffic patterns.

3.1.1.4. Adaptability

One of the basic functions of routing is to deal with topological changes and maintain reachability. When topology changes as a result of failures and repairs, a routing algorithm has to be able to rebuild the routing table automatically. When link or node failures are detected,

alternative paths have to be found quickly so that the connection may continue with minimum disruption. The ability to respond to topological changes depends on the information exchange. A tradeoff must be made between the speed of adaptive action and the routing overhead. Routing algorithms can not change faster than relevant information can be propagated to the decision points.

3.1.1.5. Optimality

The ultimate goal of a routing algorithm is to achieve optimal resource sharing. The quality of a routing algorithm is determined by both the satisfaction of individual users and the efficiency of the network resource utilization. A routing algorithm should produce routes that meet the individual requirements of the users and in the mean time take into account the global requirements of the network. An optimal routing algorithm should coordinate with admission control and congestion control mechanisms to achieve effective resource allocation and traffic control.

3.1.2. Decomposition of Routing Algorithms

Routing as a complex decision making procedure has many different but related functions. A routing algorithm has to monitor the network status and collect information which routing decisions can be based on. The collected information should then be propagated over the network in a timely fashion. The routing table can then be produced for all destinations in the network and finally it has to forward packets to the next hop along the route.

Four functions of the routing algorithm are discussed in the following sections: distance estimation, information propagation, route computation and packet forwarding.

3.1.2.1. Distance Estimation

A routing algorithm has to make routing decisions based on the current state of the network. It has to continuously monitor and collect information to maintain the database up-to-date.

One node may collect information about the network state by

- measuring local information that the node has direct access, eg. output queue length, link utilization.
- receiving updates from other nodes which contains explicit remote information such as delay, queue length.

- learning implicitly from the packets it receives from other nodes.

The frequency at which the information is updated is important. Highly frequent updating may improve the accuracy but it may also cause a substantial amount of overhead. The route updating period has to be decided according to the network environment.

The actual costs of the links or the *link distances* depend on the metrics used by the routing algorithm. The metrics provide the criteria by which the routing algorithm makes the selection. Some possible metrics are hop count, delay, throughput, link utilization or any combination of these. The choice of metrics has a great impact on the quality of routes and the stability of routing. It determines whether a routing algorithm can adapt to traffic changes and the speed of adaptation. For example, hop count metric can only reflect changes in topology while delay and queue length etc can sense changes in both topology and traffic load. In a network which has different types of services, the routing algorithm may have to provide several metrics so that different requirements can be met.

3.1.2.2. Information Propagation

Information about changes in network topology and traffic load has to be propagated to other nodes so that adjustments can be made. The procedure of information propagation must meet two basic criteria: high efficiency and high reliability.

Efficiency is essential as route updating has to carried out immediately after a change is detected yet the overhead has to be minimized. In a large network, route updating could consume substantial amounts of bandwidth. Routing updates should be generated only when a significant amount of change has been detected and the size of update should be minimized. The updates can be generated in a periodic or event-driven fashion.

Reliability means that updates must be transmitted successfully in the face of network failures and treated correctly after failure recovery. It is important that all the nodes hold consistent information otherwise long-lasting loops can be formed. The routing algorithm must also deal with the updates out of sequence and discard duplicate or delayed updates.

3.1.2.3. Route Computation

The process of route computation is the heart of the routing algorithm. It determines the best routes for traffic through the network based on the information collected so far. Shortest-path algorithms have been widely used in route computation in which the routing algorithm

attempts to optimize the performance by minimizing the distance of the route. The routing decisions can be made in many different ways. In a fixed routing algorithm, the computation might be done off-line and fixed for a relatively long time. On the other hand, an adaptive routing algorithm may update its routing table whenever significant changes are detected. In an adaptive routing algorithm, the route computation can be carried out in a centralized or distributed fashion. We will discuss these in more detail in the next section.

The other important function of route computation is to support Quality of Service (QoS). Traffic from different applications may have different requirements in priority, delay, bandwidth, reliability, security and cost. To provide such services, route computation must derive several routes with different performance attributes.

3.1.2.4. Packet Forwarding

The packet forwarding procedure is relatively simple when compared to other functions. The route computation results in a routing table. In many single path routing algorithms, the node looks up the destination in the routing table to obtain the the number of the output line and forwards the packet to the next hop. In some algorithms, there may be more than one route for a destination. The node has to select one route based on some pre-specified criteria. For example, in a multipath routing algorithm, packets are forwarded to several output lines according to certain probabilities.

3.1.3. Classification of Routing Algorithms

Routing algorithms can be classified in many different ways depending on the criteria used. The two most fundamental features that distinguish the routing algorithms are the way in which the routing decisions are made and the frequency of the routing decisions. We now discuss several basic classes of routing algorithms and examine their advantages and disadvantages in some detail.

3.1.3.1. Centralized and Distributed

There are two basic approaches in making the routing decisions: centralized and distributed. In a centralized routing algorithm, the routing decision are made only in one or a few centres and distributed to each node in the network. In a distributed routing algorithm, all nodes participate the process of decision making.

The algorithms for the centralized computation are well known. Many algorithms in graph theory can be applied directly. In a centralized algorithms, only the centres are responsible for route computation, therefore the overhead is reduced on other nodes and sophisticated algorithms such as the disjoint multi-path algorithm can be used. However, there are also many disadvantages in centralized routing. One serious problem is the vulnerability of the centres. Since critical functions such as routing decision making are carried out only at the centres, any failures of the centres may lead to catastrophic results. Different routing mechanisms have to be used to ensure the reporting and updating of routing information. When a node detects a change, the current route to the centres may not be available if it is affected by the latest change. Also, due to the delay in reporting changes and updating routing tables, the new routing table may not be distributed reliably to all the nodes in the network by the current routing tables. In a large network, the computation may take unacceptable time even on a substantial CPU and the route updates can also consume large amount of bandwidth. The routing traffic is heavily concentrated on the lines leading to the centres. The resultant heavy load and possible congestion make the centres more vulnerable.

Most of the modern routing algorithms fall into the distributed class where routing decisions are made in a distributed fashion. The most important advantage of distributed routing is its high survivability in the face of link or node failures. When network components fail, the network can still adapt to a new topology. Even in the extreme circumstances when the network is partitioned into several separate networks, each of them may still operate properly. Distributed computation also reduces the amount of information that has to propagate. However, distributed routing computations are usually more complex. The routing algorithms must ensure that the distributed status information and routing tables are consistent among all nodes, otherwise long-lasting routing loops may formed which may have severe effects on routing performance. There are two basic classes of distributed routing algorithms widely used in today's networks: *distance-vector* algorithms and *link-state* algorithms. In a distance-vector algorithm, the process of route computation is carried out in a distributed way with each node performs part of the computation. In a link-state algorithm, the information of the link states is distributed to all nodes and the routing tables are calculated independently on all nodes. We will discuss the two classes of algorithms in detail later.

3.1.3.2. Static, Quasi-Static and Dynamic

One of the essential requirements of the routing algorithms is that they must be able to adapt automatically to topological changes and possibly traffic changes as well. The frequency at which the routing updates the routing tables is an important characteristic which reflects the adaptability of the routing algorithms. According to how frequently the routing tables are updated, routing algorithms can be classified as *static*, *quasi-static* or *dynamic*.

In a static routing algorithm, the choice of routes is predetermined and fixed for a relatively long time period (months or even years). The routing tables are often calculated off-line based on the statistics of the traffic load and then fed into nodes. Static routing algorithms are extremely simple and can be easily implemented. However, as they are not able to adapt to topological and traffic changes, the networks with static routing algorithms are very vulnerable to resource failures and traffic changes. In practice, static routing algorithms often have an alternate route for each destination in case the main route fails. Static routing algorithms are often found in small networks with stable topology and traffic load.

A dynamic routing algorithm, in contrast, allows continuous changes in routing decisions to reflect the current traffic and topology. Truly dynamic routing can be very difficult to achieve as it requires large amount of status information exchanges and route computation which may result in unacceptable overhead. Optimal routing decisions rely on up-to-date details of traffic load which may not be possible to obtain due to the propagation delay. Moreover, it is also difficult to have an accurate estimation of the traffic load within very short time period.

The routing algorithms used widely in today's networks fall approximately into the quasi-static category, in which the routing tables remain unchanged for short periods of time but can be updated when significant changes occur. In a quasi-static routing algorithm, each node measures the link distance over a period of time and generates updates if significant changes are detected. This period, called the *route updating period*, is an important parameter of the routing algorithm. The route updating period has to been chosen based on the tradeoff between adaptability and overhead. The adaptability of the routing algorithms also depends on the link metrics that determine that way in which the link distances are calculated. Some quasi-static algorithms adapt both to topological changes and traffic changes while the others only to topological changes. In the next section, we will discuss in more detail two widely used quasi-static algorithms.

3.1.4. Distributed Shortest-Path Routing Algorithms

The routing algorithms that are most widely used in today's computer networks belong to the class of quasi-static distributed shortest-path routing algorithms. In such algorithms, each node attempts to route packets to their destinations over paths of minimum distance and updates the distances periodically to adapt to topological and traffic changes. There are two main groups of algorithms: *distance-vector* algorithms and *link-state* algorithms. In a distance-vector algorithm, each node maintains a routing table containing the distance of the shortest path to every destination in the network. A node only informs its immediate neighbors of any distance changes to any particular destinations. Examples of distance algorithms include the old ARPANET routing algorithm [McQu77], the NETCHANGE algorithm of the MERIT network [Taji77] and cisco's IGRP [Hedr89]. In a link-state algorithm, each node keeps track of the entire network topology and computes the routing table based on the link distance information broadcast by every node in the network. Link-state algorithms have been used in the new ARPANET routing protocol (SPF) [McQu80], OSPF [Moy89] and ISO's IS-IS [Orga89].

3.1.4.1. Distance-Vector Algorithms

The distance-vector algorithms are based on an algorithm developed by Ford and Bellman [Ford62]. The idea is to compute the shortest paths from every node to every other node by repeating a distributed version of Ford-Bellman algorithm. The algorithm requires very little information to be stored at the network nodes. In fact, it is suffices for a node to know the distances of its outgoing links.

In the original Ford-Bellman algorithm, the problem is to find the shortest paths from a source node to all other nodes. The trick is to first find the shortest path lengths subject to the constraint that the paths contain at most one arc, then to find the shortest paths within at most two arcs, and so forth.

Let the $D_i^{(h)}$ be the shortest path length from source node 1 to node i, subject to the constraint that the path contains at most h arcs. We take $D\{^{(h)} = 0$ for all h. Let d_{ij} be the length of between the adjacent node i and node j and $d_{ij}=\infty$ if the (i,j) is not an arc of the graph. The Ford-Bellman algorithm can be written as:

Initially, $D_i^{(0)} = \infty$, for *all* $i \neq 1$

For each successive $h > 0$, $D_i^{(h+1)} = min\,[D_j^{(h)} + d_{ji}]$, $i \neq 1$

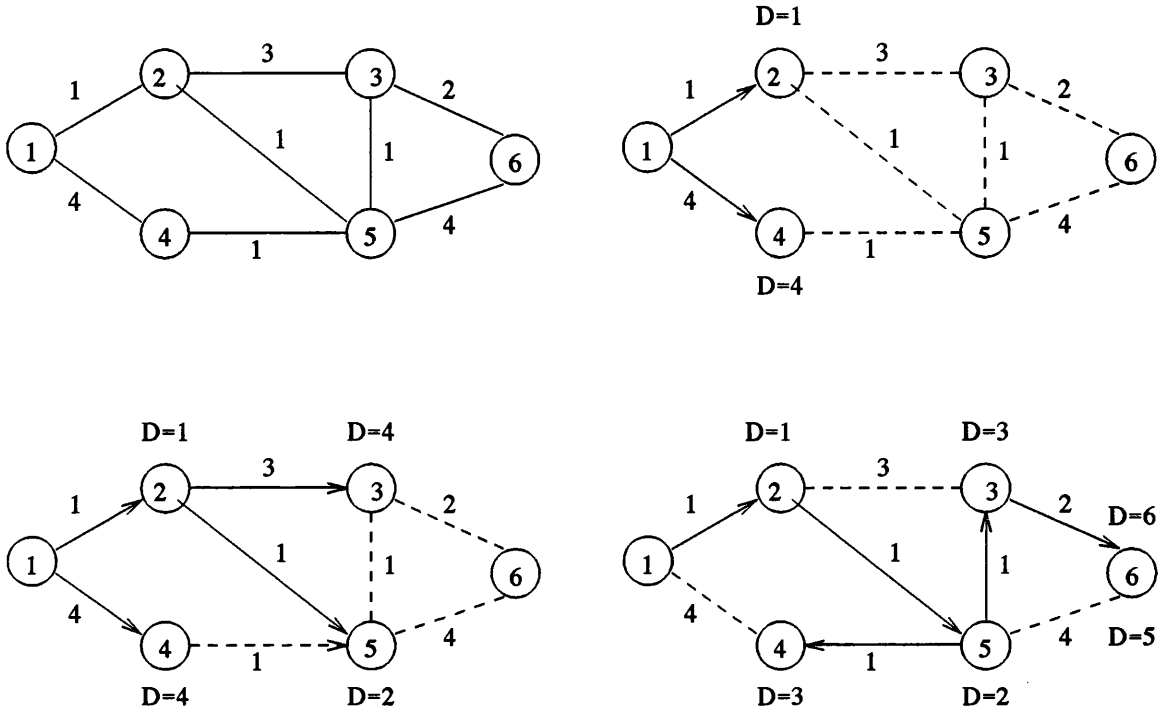


Figure 3.1: Ford-Bellman Algorithm

The Ford-Bellman algorithm is illustrated in Figure 3.1. The actual equations used in distance-vector algorithms are equivalent but slightly different from the original Ford-Bellman algorithm. In distance-vector algorithms, we focus on the shortest distance D_i from each node i to a destination node (node 1). The equations are given by $D_i = min\,[d_{ij} + D_j]$, $i \neq 1, j \in N(i)$ and $D_1 = 0$ where N(i) denotes the set of current neighbors of node i.

The modified algorithm is well suited for distributed computation since each iteration of the algorithm can be executed at each node i in parallel with other nodes.

In practice, each node in the network measures the distance of its outgoing links. When it detects significant changes in d_{ij}, it updates the distance D_i and informs its neighbors the new distance. Upon receiving new distance information from a neighbor, a node executes the iteration

$$D_i := min\,[d_{ij} + D_j],\ j \in N(i)$$

using the latest estimates D_j received from the neighbors $j \in N(i)$ and the latest distances of its outgoing links.

This algorithm is still valid when executed asynchronously as described above. If no more changes occur after a number of link length changes, the asynchronous algorithm can find the

correct shortest paths of every node i.

In some cases, however, the convergence can be very slow and an excessive number of messages are needed to complete the shortest path search. One well-known problem is called the *Count-to-infinity* Problem.

In the distance-vector algorithms, nodes do not have complete topology information. When link distance changes, the algorithm has to update the routing table by recomputing the shortest paths over the entire network. Before the computation is completed, the routing table may not be consistent and loops may be formed.

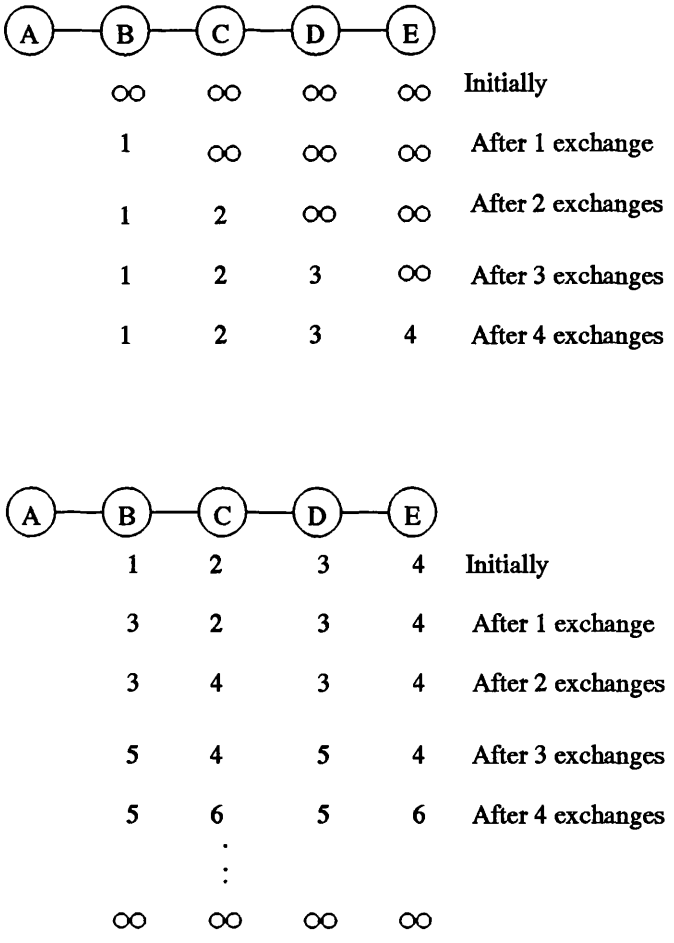


Figure 3.2: An Example of Count-to-Infinity

The distance-vector algorithms have the property of reacting quickly to good news but slowly to bad news. Consider the simple subnet of Figure 3.2, where delay metric is the number of the hops. Initially, node A is down and all other nodes have a delay of infinity to A. When A

comes up, node B learns that it has a delay of one hop to A. At the next exchange of routing information, node C learns from B that A is two hops away. In such a way, the information about node A will reach node E in four exchanges. Now consider the situation of all nodes initially up, with A suddenly crashing. At the first information exchange, B does not hear anything from A, but C claims that it has a path of two-hop to A. Now B thinks that it can reach A via C with a length of three hops. In the next exchange, however, node C hears from B that its distance to A has increased to three hops. Node C therefore changes the entry for node A to four hops. As has been shown in Figure 3.2, node B only increases the entry for node A to five hops after four exchanges. The infinity is often set to the longest path + 1, so that all node will eventually learn that node A is down as the distance approaches a large number. During the period of count-to-infinity, loops may be formed.

The count-to-infinity may occur only if a routing update received by a node is based on an update which previously was sent by that node. In distance-vector algorithms, when a node receives an update about a particular destination, the good news (reporting a shorter distance) will replace the bad news (reporting a longer distance). If a link changes from a shorter distance to a longer distance, the bad news will be suppressed by updates based on the good news sent previously. It is clear that some measures have to been taken to limit the effects of the obsolete good news and speed up the propagation of the bad news so that long-lasting loops can be avoided.

In the last several years, there have been a number of solutions proposed to the looping problems of distance-vector algorithms. Recent work by Merlin-Segall, Jaffe-Moss and Garcia have achieved loop-free routing in distance-vector algorithms [Merl79, Jaff82, Garc89a].

One of the techniques that has been used is called *split-horizon*, with which a node never chooses as its next-hop towards a destination a neighbor that has reported the node as its own next-hop to the same destination. Stern [Ster80] proposed an algorithm that uses the split-horizon technique, in which a node sets it distance to a destination to infinity when the distance to a destination is increased. Garcia [Garc88] also proposed a similar algorithm, in which a node chooses, as the next-hop to a destination, a neighbor that has reported either a decreasing distance to that destination or a constant distance and a constant next-hop. Another technique that has been used is called *hold-down* [McQu74]. With this technique, when a node receives an update from its current next-hop indicating a distance increase, it must wait for a fixed period of time before making any changes to its routing table. However, none of those techniques solve all the looping problem in distance-vector algorithms.

There are several algorithms that have been proposed to ensure loop-free routing by means of internodal coordination. The algorithm by Merlin-Segall [Merl79] is based on the propagation of information with feedback along a tree rooted at the network destination and spanning all network nodes. Jaffe-Moss [Jaff82] further developed this idea by ensuring internodal coordination when propagating routing information. In Jaffe-Moss' scheme a node is prevented from receiving old good news from itself by not allowing that node to accept any good news about a destination until every node uptree from it has received the bad news. All these algorithms do not prevent looping in all cases. These algorithms can also be viewed as using a technique called *diffusing computation* introduced by Dijkstra and Scholten [Dijk80] to check the termination of a distributed computation. The diffusing computation grows by sending *queries* and shrinks by receiving *replies* so that the process starting the computation is informed when it is completed and there is no false terminations. More recently, Garcia [Garc89b] presented a new algorithm that generalizes these algorithms on diffusing computations and achieves loop-free routing.

3.1.4.2. Link-State Algorithms

Due to the problem of looping in distance-vector algorithms, a different class of algorithms called *link-state algorithms* have been proposed. In a link-state algorithm, each node has complete information of network topology so that each node can carry out route computation independently. When a node detects any changes in link distances, it sends out an update to all other nodes by broadcasting. Upon receiving an update, each node then re-calculates the routing table independently. Routing loops may exist during the updating period but routing tables eventually become consistent when each node has updated its routing table.

The shortest path algorithm used in link-state algorithms is one developed by Dijkstra [Dijk59]. Its worst-case computational requirements are considerably less than those of the Ford-Bellman algorithm. The basic idea is to find the shortest paths in order of increasing path length. We view each node i as being labeled with an estimate D_i of the shortest path length from node 1. When the estimate becomes certain, we regard the node as being permanently labled, which is kept in a set P. The node added to P at each step will be the closest to node 1 out of those that are not yet in P.

The algorithm is illustrated in Figure 3.3 and detailed as follows:

Initially $P=\{1\}$, $D_1=0$, and $D_j=d_{1j}$ for $j\neq 1$.

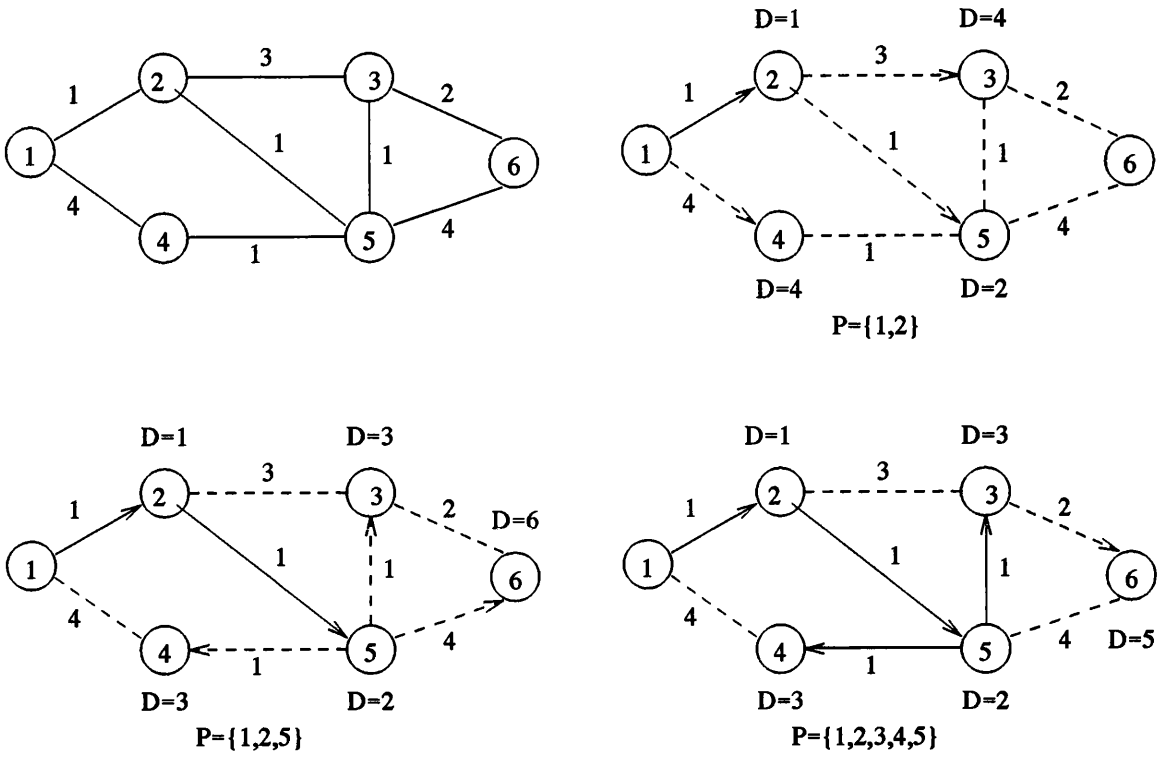


Figure 3.3: Dijkstra Algorithm

- Step 1: finding the next closest node.

  Find $i \notin P$ such that $D_i = \min D_j, j \notin P$

  $P = P \cup \{i\}$.

  If P contains all nodes then terminate. Otherwise

- Step 2: updating the labels.

  For all $j \notin P$, $D_j = min[D_j, D_i + d_{ij}]$

  Goto Step 1.

The Dijkstra algorithm is used for calculating shortest paths of a static topology. When used in computer networks, temporary routing loops may exist during the routing update period when the new routing information is in transit and temporary inconsistency of the routing information exists. Routing algorithms based on the Dijkstra algorithm often use flooding to propagate information, which is fast and robust. The measure proposed by Garcia [Garc89] can also be used for link-state algorithm to eliminate temporary routing loops.

3.1.5. Source Routing

In the routing techniques discussed above, the function of routing is provided by intermediate systems (nodes, gateways etc). End systems only need to know the first-hop that they forward the packets to. In many cases, end systems simply send packets to the default first-hop and the packets will then be forwarded by intermediate systems. In many networks, however, the end systems can also dictate a path through the network. This technique is called *source routing*. With source routing, the end systems must have information about the network topology and specify a list of addresses in the packet which it should follow.

There are two basic forms of source routing [Post81]. One form, called *strict source routing*, includes a complete list of addresses that the packet must follow to reach the destination. The packet will be discarded if the source route can not be followed. The other form, called *loose source routing*, also includes a list of addresses that the packet should follow but it allows multiple network hops between successive addresses on the list.

Source routing provides a useful way of exercising control and overruling the current routing decisions by the network. It can be used in network monitoring, fault isolation, security enforcement and protocol experiments.

3.2. Congestion Control

Congestion occurs when the traffic demands exceed the resources at a bottleneck. If the congestion is persistent, the excessive packets accumulating at the bottleneck may lead to buffer overflow and even deadlock in some circumstances. Congestion control attempts to maintain fair and optimal resource sharing at the bottleneck. In some literature, *congestion control* is used to refer to congestion recovery mechanisms while *congestion avoidance* is used to refer to congestion prevention mechanisms. The term *flow control* is also widely used, especially when referring to mechanisms for preventing buffer congestion at the end systems. In this dissertation, we use the term *congestion control* for all those mechanisms that deal with congestion at the network switches and the term *flow control* for those that deal with congestion at the destination buffer.

The functions of a congestion control scheme can also be divided into two levels. At the basic level, a congestion control scheme has to maintain an optimal operating point at the bottleneck. At a higher level, a congestion control scheme has to ensure fairness among users sharing the bottleneck resources. Congestion control operates in a much smaller time scale than routing does. The unpredictable nature of the traffic sources and the delay in feedback

information present the major problems for congestion control.

In this section, we first discuss the design goals, composition and classification of congestion control schemes, and then examine two widely used end-system congestion control schemes in detail.

3.2.1. Design Goals

3.2.1.1. Efficiency

Congestion control schemes have to be efficient since the resources available for congestion control are very limited. During the congestion there is a shortage of resources, high overhead may have adverse effects on the congestion. When congestion does occur, the build-up of packets at the bottleneck is often very rapid, so a congestion control scheme must be able to respond quickly and take immediate action.

3.2.1.2. Optimality

An important decision that a congestion control scheme must make concerns what is the optimal traffic load that the scheme attempts to maintain at a bottleneck.

When traffic is light, throughput increases as the traffic load increases. After the network pipe of the links has been filled up, the packets start to accumulate in the buffer of the bottleneck. The queuing delay increases rapidly if the traffic load increases further. If the traffic load is so high that the buffer is overflowed, the packets have to be discarded. The end system has to retransmit the lost packets and the effective throughput starts to decline. The path is said to be *congested.*

It has been suggested that the network should operate at a point where the *resource power* (throughput/delay) is maximized [Rama88]. This point is near the knee of the throughput-delay curve and the queue length is around one. However, due to the statistical nature of traffic and the propagation delay in the feedback, it is not possible to maintain network operating at the knee without any loss of bandwidth.

Keshav [Kesh91a] proposed another definition of optimality: a flow is optimal if in every time interval $[t_1,t_2]$, there are no buffer overflows and there is no loss of bandwidth at the bottleneck. This definition allows a wider range of traffic load at which a flow may operate.

The buffer in the bottleneck is used to accommodate the fluctuation of the traffic. Therefore the average buffer occupancy should be maintained at a point where there is no loss of bandwidth, in the meantime the queuing delay is minimized. From the users' point of view, this operating point enables the bottleneck to operate at the maximum capacity while keeping the queuing delay low. From the bottleneck's perspective, the maximum amount of free buffer has been allocated for traffic bursts while a minimum number of packets has been stored in the buffer to fill the gaps in the flows.

In practice, it may be difficult to maintain a traffic flow exactly at such an optimal operating point. An operating point with a reasonable length of queue size should be regarded as acceptable.

3.2.1.3. Fairness

When traffic demands exceed the available resources, congestion control schemes must ensure not only that traffic demands are reduced but also that each user has an equal share of the resources.

Fairness may be the most challenging problem for congestion control in a datagram network. The difficulty is that neither the end users nor the intermediate systems have complete information on the network states. With such a restricted view, it is difficult for the end users or the intermediate systems to take precise control actions.

An end user can derive information about overall traffic load from the round trip delay. However, it is unable to determine whether the delay is caused by the constraint of network resources (eg. link bandwidth, switch speed) or by sharing with other users.

An intermediate system can also gather information on traffic load by monitoring the queue length and link utilization. However, it can not figure out the contribution of a particular user to the traffic. To do so requires each connection to have an unique identification and each intermediate system to monitor traffic on the connection-by-connection basis.

3.2.1.4. Stability

A congestion control scheme should converge to a fair and optimal point provided there are no changes in the traffic load with minimum oscillation. To achieve this, some degree of coordination of all the users sharing the same bottleneck may be necessary. Self-optimization schemes may lead to divergence [Jain89].

The propagation delay in the information feedback also presents a problem to the congestion control schemes. It usually takes more than one round trip time for the sender to detected changes in the network conditions. The effects of a control action can not be assessed by the sender for another round trip time. Therefore, transient information, which may be obsolete before an adjustment can be made by the end system, should be avoided.

3.2.2. Decomposition of Congestion Control Schemes

A congestion control scheme must have two basic components: *a monitor component* and *a control component.*

The monitor component is responsible for collecting state information on the network conditions and provides feedback to the control component. The nature of the collected information in some way determines the control actions that can be taken. If the information is binary, the control component has to adjust traffic recursively towards the optimal operating point. If the information is quantitative, the optimal operating point may be approached more quickly.

The control component actually initiates the remedial action to adjust the traffic load. There are two basic actions that can be taken when congestion occurs. An end-system can reduce its traffic rate so fewer packets enter into the network. An intermediate-system can also discourage the users who send excessive packets by discarding their packets.

Network Layer	Transport layer
• Connection mechanism	• Round trip delay estimation
• Packet queueing and service algorithms	• Timeout estimation
• Packet discarding policy	• Retransmission policy
• Routing algorithms	• Acknowledgement policy
• Buffer allocation policy	• Packet caching policy

Table 3.1: Mechanisms Related to Congestion Control

The monitor component and control control can be either in the end-systems or in the routers. An end-system can derive the state information from the acknowledgements while a router can monitor its resource consumption directly.

Congestion control schemes are also affected by many other transport layer and network layer mechanisms. Table 3.1 lists some of them.

3.2.3. Classification of Congestion Control Schemes

3.2.3.1. End-System Control and Network Control

Depending on where the control actions are taken, congestion control schemes are divided into two groups: *end-system control* and *network control.* The two types of control are in fact complementary and they form a complete congestion control system. Under congestion, the end-system control actually reduces the traffic at the sources while the network control maintains a fair allocation of the resources and therefore discourage ill-behaved users from sending more packets. An end-system scheme requires one round trip time for the congestion to be detected and another round trip time for the control actions to become effective. In contrast, a network control scheme can take immediate actions at the point where the congestion occurs. Nevertheless, the traffic can not be reduced without adjustment at the sources.

3.2.3.2. Window Control and Rate Control

There are two basic approaches in end-system congestion control towards regulating the traffic demands: *window control* and *rate control.* In a window control system, the source regulates its traffic demand by changing the window size, ie. the packets in-transit. In a rate control system, the source controls the transmission rate explicitly.

Window mechanisms were originally used for managing buffer congestion at the destination and were later adopted for controlling network congestion using feedback information. Combined with sequencing, acknowledgement and retransmission, window mechanisms form a complete framework for reliable data transmission. The advance of the window boundaries are driven by acknowledgements therefore window mechanisms are self-clocking and very easy to implement. The window size in fact represents the total packets in the network, ie. the packets in-flight on the links and the packets queueing at the switches. Therefore the window size has a simple relation with the queue length. When the queue length is large than zero, any changes in the window size is directly reflected in the queue length.

However, in window mechanisms, the transmission rate depends on not only the window size but also the round trip delay. Enforcement of fair sharing of bandwidth resource is much

more difficult than rate control mechanisms. In most window control schemes, the users with longer round trip delay often get less than the fair share of the bandwidth. Because the transmission of packets is driven by the acknowledgements, packets may be sent out back-to-back, resulting in bursty traffic patterns and increase in the queue length fluctuation.

Fair sharing of network resources can be achieved more effectively by rate control mechanisms since they are independent of round trip delay and most resources at the switches are usually allocated in terms of service rate or bandwidth. In rate control schemes, packets are sent out in a more smooth fashion and therefore queue length fluctuation can be reduced. Most rate control mechanisms have two parameters: burst size and burst rate. By changing those two parameters, rate control schemes may accommodate different requirements of traffic sources.

Rate control schemes have to maintain the inter-packet or inter-burst gaps to ensure proper transmission rate. The transmission of packets has to be triggered by timers, which can be very expensive in high speed networks. Sequencing and error recovery are not covered by rate control mechanisms therefore they have to be carried out separately. The relationship between the transmission rate at the source and the queue length at the switches depends on the nature of the traffic and can not be formulated in a simple way. At high traffic load, small changes in the transmission rate may result in large queue length fluctuation.

3.2.4. End-System Congestion Control Schemes

In this section, we examine two widely used congestion control schemes. The binary feedback scheme developed by a group of people at DEC [Rama88] is used in DEC's Digital Network Architecture and has been adopted for ISO TP4 as well. The slow start and congestion avoidance algorithm was developed by Jacobson [Jaco88] and implemented in UNIX BSD 4.3 tahoe release.

3.2.4.1. DEC Binary Feedback Scheme

The DEC binary feedback scheme is based on an explicit feedback signal. Each packet has a bit called the *congestion indicator* in its header. When a switch is congested, it sets the congestion indicator bit. Any switches that are not congested just ignore the congestion indicator bit. When the packet arrives at the destination, the congestion indicator is copied into the header of the acknowledgement packet. This acknowledgement packet then brings the congestion indicator back to the source.

The feedback signal attempts to maximize the *Power* which is define as *(throughput/delay)* [Rama88]. In practice, the congestion indicator is set when the queue length is greater than or equals to 1. The queue length is calculated by using moving average over each cycle starting from the regeneration point of the queue length. When approximately two window worth of acknowledgement packets are received, a decision is made as whether to increase or decrease the window size. If 50 percent of the acknowledgement packets are received with the congestion indicator set, the window size is decreased. Otherwise, the window size is increased.

The algorithm used to adjust the window size is called *additive increase and multiplicative decrease (AIMD)* [Chiu89] which can be represented as follows:

$$\textit{Increase}: \quad W_c = W_p + b, \; b \geq 0;$$

$$\textit{Decrease}: \quad W_c = dW_p, 0 < d \leq 1.$$

The idea is that each user gains the same amount in the additive increase phase and loses in proportion to the share it has in the multiplicative decrease phase. Therefore the users who have larger than fair shares of the bandwidth lose some advantage in each iteration. Eventually all users have equal shares.

There are other two congestion control schemes [Jain89, Jain86], also proposed by Jain at DEC, that use the AIMD algorithm for window adjustment. However, they are based on timeout and delay gradient respectively for congestion signalling.

There are two problems associated withe the DECBIT scheme. First, there is a bias against the connection with longer round trip times. Since it is the window size that is adjusted, the window sizes of all connections will eventually become equal. Obviously, the connections with longer round trip times have lower throughput. Second, the convergence of the algorithm is slow. In the DECBIT scheme, the actual parameters b and d are 1 and 7/8 respectively. The frequency of decision making is once every two round trip times so each adjustment needs at least two round trip times. Because of the increase and decrease adjustment, the number of operations needed to reach convergence can be very large. By using a larger b and a smaller d, the adjustment can be speeded up but this will increase the magnitude of the oscillation when the convergence is reached.

3.2.4.2. Slow Start and Congestion Avoidance Algorithm

The congestion control algorithm embedded in the 4.3-Tahoe BSD TCP implementation was developed by Van Jacobson when the Internet was experiencing congestion collapse [Jaco88]. The algorithm has dramatically improved congestion control over the Internet and is now regarded as an Internet Standard.

Jacobson's algorithm consists of two separate window adjustment algorithms: *slow start* and *congestion avoidance*. Without the slow start algorithm, TCP always sends a full window's worth of back-to-back packets to the network when it starts initially or restarts after a timeout. Such bursts of packet trains often cause a large overshoot at a slow bottleneck and result in packet losses. The retransmission of the lost packets may lead to another overshoot. To avoid overwhelming a slow bottleneck with bursts of packets, the slow-start algorithm sets the congestion window *cwnd* to 1 (in units of the maximum packet size) and then opens the window exponentially to the slow start threshold *ssthresh*. The congestion avoidance algorithm uses timeout as congestion signal and use the AIMD algorithm for window adjustment. Jacobson's slow start and congestion avoidance algorithm is very similar to the CUTE algorithm [Jain86].

The slow start and congestion avoidance can be combined into one algorithm although their purposes are quite different. When a timeout is detected, the threshold *ssthresh* is reduced to half of the current window size and then *cwnd* is reduced to 1. During the slow-start phase, each acknowledgement increases *cwnd* by 1. After *cwnd* reaches *ssthresh*, *cwnd* is increased by $\frac{1}{cwnd}$ for each acknowledgement received. The actual C code of the combined algorithm is simple:

When a new connection is set up, the sender does

```
cwnd = 1;
ssthresh = maxwnd;
```

When an acknowledgement is received, the sender does

```
if (cwnd < ssthresh)
  cwnd += 1;
else
  cwnd += 1/cwnd;
```

When a timeout is detected, the sender does

```
ssthresh = max(min(cwnd, wnd)/2, 2);
cwnd = 1;
```

The actual sending window size *wnd* should not exceed the receiving size advertised by the receiver *maxwnd* therefore the sender always does

```
wnd = min(cwnd, maxwnd);
```

Simulation studies have shown that after initial adjustment the above algorithm exhibits clear oscillatory patterns in sending window size, round trip time and bottleneck queue length. [Shen90, Zhan89]. This behavior is caused by the congestion avoidance algorithm. When a timeout is detected, the sending window is set to 1 and opens up exponentially to half of the window size before the adjustment. The window then increases linearly by approximately 1 for each round trip time. Eventually new packet losses result in another timeout and this process repeats. Oscillation of such high magnitude in sending window size inevitably leads to drastic round trip time and queue length fluctuation. The periodic packet losses at each peak of the fluctuation have severe impact on the network performance. Since the retransmission timeout estimate is usually longer than one round trip time, the pipeline of the packet stream is already broken when the timeout is detected and the traffic flow has to be re-started gradually to avoid overwhelming the bottleneck.

While window oscillation is the very measure used in the AIMD algorithm to probe the network conditions [Chiu89], the oscillation seen in Jacobson's algorithm is aggravated by the periodic packets losses. We believe that by eliminating the periodic packet losses we can substantially reduce the magnitude of the oscillation and improve the performance.

Now let us examine the traffic changes from the perspective of a particular user A. Suppose that over a time period, the traffic load at the bottleneck increases by T. From A's perspective, the traffic increase can be divided into external traffic increase T_e and internal traffic increase T_i, ie. $T = T_e + T_i$. The external traffic increase T_e created by other users (eg. a new user joins in) is unpredictable and beyond A's control. User A can reduce its traffic when it detects such increase. Nevertheless, due to the feedback delay, packet losses can occur if such increase is rapid. On the other hand, internal traffic increase T_i caused by user A itself is controllable. User A should be able to avoid overloading the bottleneck when there is no external traffic increase. An extreme example is that there is only one connection over the path.

The AIMD algorithm has two adjustment phases. When the traffic load is steady or decreasing, the algorithm is in the probing phase. The internal traffic increases linearly to absorb released resources and to probe the state of the bottleneck. When a congestion signal is received, the algorithm enters into the reduce phase and reduces its traffic proportionally. In Jacobson's algorithm, packet losses and its subsequent timeouts are used as the congestion signal. As a result, packet losses inevitably occur at the peak of each increase phase. We believe that by using a more subtle congestion signal we can eliminate the packet losses caused by internal traffic increase. When a timeout is detected, Jacobson's algorithm decreases the slow-start threshold to half of the current window. Such dramatic reduction can speed up the adjustment when there are substantial traffic changes. On the other hand, it leads to high magnitude of oscillation during normal resource probing.

3.3. Summary

The characteristics and classification of routing algorithms and congestion control schemes are discussed. Three aspects are examined in detail: design goals, decomposition and classification.

Two shortest path routing algorithms and two end-system congestion control schemes which are widely used in real networks are described and their problems are analyzed.

CHAPTER FOUR

ANALYSIS OF SHORTEST PATH ROUTING ALGORITHMS

This chapter looks at the behavior of shortest path routing algorithms in a dynamic environment. The shortest path routing algorithms are examined from the perspective of control theory and decision making and six major problems are identified and analyzed in detail.

4.1. Introduction

Shortest-path routing algorithms have served remarkably well in the network environment where traffic load is light and network conditions change slowly. Shortest-path algorithms are able to respond to topological changes automatically and adjust routing decisions when traffic changes. In the presence of congestion, shortest-path routing algorithms can reduce the traffic away from the overloaded paths.

As the networking speed increases, the range of traffic rates which the network has to deal with widens. Mixture of different traffic sources such as video, voice and data makes the traffic pattern less predictable and its distribution more uneven.

In a network environment where traffic approaches the capacity of paths and changes dynamically, shortest-path routing algorithms, particularly those that attempt to adapt to traffic changes, frequently exhibit instability, derive poor-quality routes and result in performance degradation [Bert82, Khan89].

4.2. Overview

As discussed in the previous chapter, a routing algorithm has four basic functions: distance estimation, information propagation, routing computation and packet forwarding.

From the control-theoretic point of view, distance estimation belongs to the problem area of adaptive parameter tracking [Sore80] When traffic changes slowly, the distances of links between two consecutive route updating periods are assumed to be closely correlated, so that, based on observations of the distances in one route updating period, the distances in the next period can be estimated. The difference between the estimated value and the measured value is called the *estimation error.* When the estimation error increases, the estimated distances which the route computation is based on become less valid therefore the quality of the routes deteriorates. Due to the unpredictable nature of the traffic and the delay in feedback information, accurate estimation is extremely hard.

In shortest-path routing algorithms, the link distances are decided based on the particular routing metric used (eg. delay, queue length, throughput, hops). With traffic-sensitive routing metrics (eg. delay), each node estimates the link distances for the next route updating period by averaging the link distances in this period. For example, in the SPF routing algorithm, the delay of each link is averaged every 10 seconds and then used for calculating the routing table for the next route updating period. The underlying assumptions of this approach are that the route updating does not affect the traffic distribution significantly and the statistics of the traffic remain unchanged. As we will see in the next section, such assumptions are approximately true only when the traffic load is light.

The problem is further complicated by the interaction between the distance estimation procedure and route computation procedure. The routing decisions for the next route updating period are made based on the current estimated link distances. However, the distribution of the traffic is determined by the resultant routing decisions. This recursive nature makes it difficult to obtain accurate distance estimation with simple algorithms.

The route computation can be considered as a problem of individual decision making under uncertainty [Luce57]. When one node chooses a path, it must not only consider the delay changes incurred by its own traffic and the effects on traffic sharing the same link, but also predict the effects of routing decisions made in the same way by other nodes. With only partial information about the traffic distribution and limited resource for route computation, it is difficult for the node to determine the overall effects of its routing decisions.

In shortest-path routing algorithms, each node simply chooses the paths of minimum distances based on information collected in the previous route updating period. This strategy is based on *individual rationality* rather than *group rationality* [Cyer87]. Each individual attempts

to maximize its use of the resource with no regard to effects of its action on the other individuals. It usually works well when there is adequate resource. However, when the resource is scare, the conflict of interests dominates the result and individual users compete with each other for more resource, which often leads to overload and collapse of the system.

It is obvious that a shortest-path routing algorithm has only *one* path for a pair of source and destination nodes at any given time. In other words, no matter how many possible paths exist from source to destination, only the best path is used. We call it *the single-path restraint.*

The single-path restraint limits the maximum flow between a pair of source and destination when there are more than one paths available. When the traffic approaches the capacity of the best path, shortest-path routing algorithms become unstable and often oscillate between different paths. The nature of individual rationality only contributes more to the instability. When distances of different paths vary widely, The paths which have comparatively shorter distances may attract too much traffic and become congested while the paths reported high distances may be abandoned and become idle.

4.3. Estimation Error

Link distance estimates provide the criteria on which the routing decisions are based. The quality of the routes ultimately depends on the accuracy of the link distance estimation, no matter what route computation algorithms are used. The estimation error can be used as a measure of routing performance.

The common approach of distance estimation is to average the link distances in current route updating period and use them as estimated distances for the next period.

Let us now examine the distance estimation process of link i. Suppose that $D_i(t-T, t)$ is the measured average delay of link i during the time period $(t-T, t)$, where T is the route updating interval. The delay information $D_i(t-T, t)$ is then propagated and received by node j at time $(t+t_j)$, where t_j is the propagation delay for the information to reach node j. Node j updates the distance of link i with $D_i(t-T, t)$ and uses it as the estimated delay for the next time period $(t+t_j, t+t_j+T)$. Let $D_i(t+t_j, t+t_j+T)$ be the actual measured delay of link i during the time period $(t+t_j, t+t_j+T)$. The estimation error of link i during the time period $(t+t_j, t+t_j+T)$ is given by

$$\Delta_i = | D_i(t+t_j, t+t_j+T) - D_i(t-T, t) |$$

We now consider a G/G/1 system under heavy traffic load. The approximate average queueing delay D_q is given by [Klei76]

$$D_q \approx \frac{(\sigma_a^2+\sigma_b^2)}{2(1-\rho)\bar{t}}$$

where σ_a^2, σ_b^2 and $\bar{t}$ are respectively variance of interarrival, variance of service time and average interarrival time.

Suppose ρ_1 and ρ_2 are respectively the utilization factors during time period $(t-T, t)$ and $(t+t_j, t+t_j+T)$. The estimation error is given by

$$\Delta_i = \left| \frac{(\sigma_a^2+\sigma_b^2)}{2(1-\rho_2)\bar{t}} - \frac{(\sigma_a^2+\sigma_b^2)}{2(1-\rho_1)\bar{t}} \right|$$

Note that $\bar{t}$ is given by $\frac{1}{\mu \times \rho}$ so we have

$$\Delta_i = \mu \frac{\sigma_a^2+\sigma_b^2}{2} \left| \frac{\rho_2-\rho_1}{(1-\rho_2)(1-\rho_1)} \right|$$

Let $\Delta_o = \mu \frac{\sigma_a^2+\sigma_b^2}{2}$ and $\Delta\rho = \rho_2-\rho_1$, we have

$$\Delta_i = \Delta_o \left| \frac{\Delta\rho}{(1-\rho_1-\Delta\rho)(1-\rho_1)} \right|$$

The estimation error is a function of ρ_1 and $\Delta\rho_1$. If we fix $\Delta\rho$ and increase ρ_1 gradually, the estimation error remains very low until ρ_1 approaches to 1. The estimation error then increases sharply (Figure 4.1). This threshold behavior results from the fact that the delay changes much more rapidly under heavy traffic.

Figure 4.2 shows the normalized estimation error as a function of percentage of changes in the utilization factor. The estimation error is low when the traffic load is decreasing but rises sharply when the traffic load is increasing. The change in traffic load $\Delta\rho$ depends on the nature of traffic and the propagation delay. When the propagation delay is large, the difference between ρ_1 and ρ_2 tends to increases. In general, the estimation error is low only when the traffic load is light or decreasing.

The route updating itself can also significantly change the traffic distribution and increase the estimation error. When the measured delay is very high or very low, it is very likely that the route updating will alter the traffic distribution in the next time period therefore the measured delay will be less valid for estimating the link distance for the next route updating period.

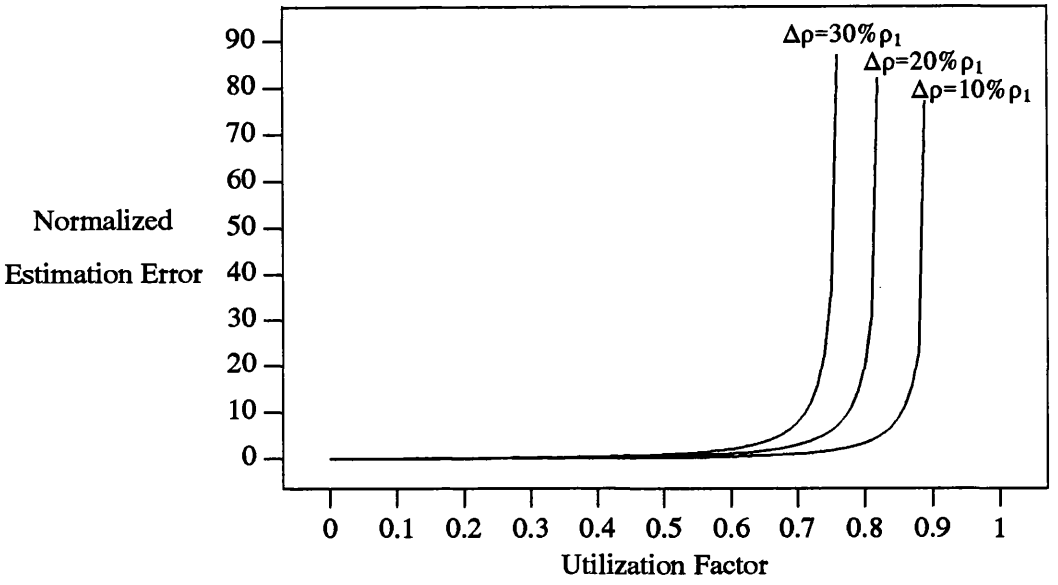


Figure 4.1: Normalized Estimation Error as a Function of ρ_1

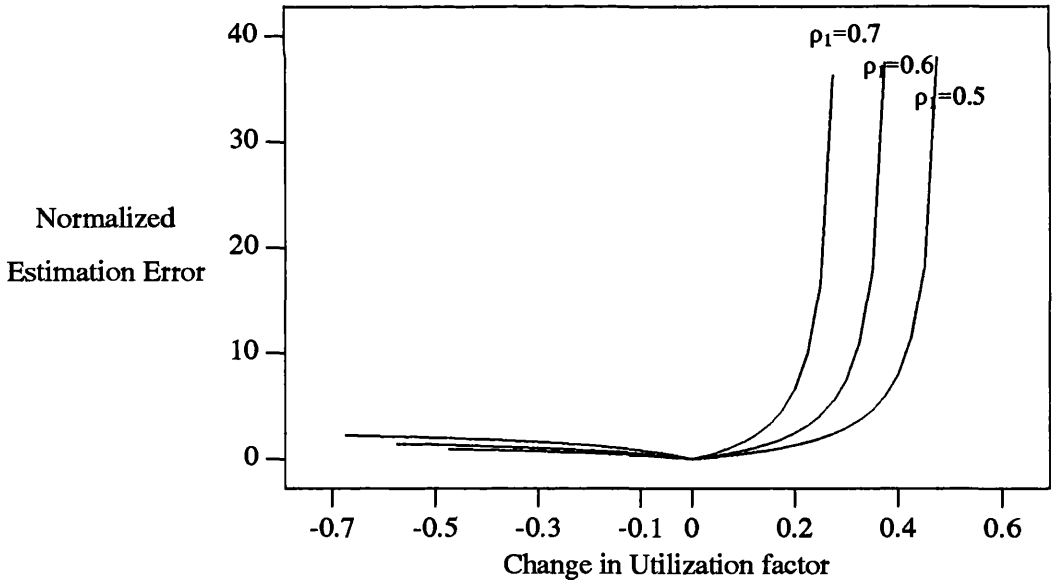


Figure 4.2: Normalized Estimation Error as a Function of $\Delta\rho/\rho_1$

The change of shortest-paths shifts traffic between different paths. When the total traffic consists of many small flows, the estimation errors resulted from traffic shifting are small. However, if the traffic is dominated by several large flows, traffic shifting can cause large estimation errors and may lead to oscillation.

4.4. Stability

Stability is of great concern in routing algorithms. Any routing algorithms should be able to reach an equilibrium in finite time, provided no continuous topological and traffic changes occur.

However, due to the single-path restraint, shortest-path routing algorithms tend to oscillate traffic flows between different paths even there are no traffic changes take place. Suppose that node x has two identical paths P_1 and P_2 to destination z. Initially they have equal distances. If during one route updating period, P_1 is chosen to be the shortest-path, the distance of P_1 rises as the traffic increases. At the next route updating point, path P_2 becomes preferred since it has shorter distance. But it will be reversed during the next route updating. The oscillation will continue and no equilibrium can be reached.

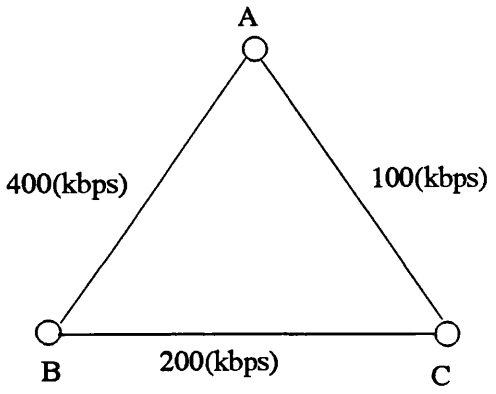
Such oscillation may be avoided to some extend by allowing route updating only when the reported distance is *significantly* different from the previous one. For example, in the SPF algorithm, a delay updating threshold is set to 64 ms. A newly measured delay is compared with the previous one. If the difference does not exceed the threshold, no updates are generated but the threshold is decreased by 12.5 ms which allows long lasting changes to be updated eventually.

Under light traffic, the distances change slowly. A small threshold can effectively damps most oscillations. But when the traffic approaches the capacity of the path, the distances increase sharply. The routing algorithms desperately attempts to search for a path which can accommodate the traffic rate. When such a path is not found, traffic simply oscillates between different paths.

The oscillation resulted from heavy traffic load has severe effects on the performance of routing. If the distance of a link between two nodes is greater than a path between the two nodes, the link will be abandoned by all the traffic. Therefore, with heavy traffic, many links may be left without any traffic while other links are severely congested.

When the traffic flow between two nodes oscillates among different paths, all traffic that shares any links on the paths are affected. In some circumstance, the oscillation can be propagated like a wave across the network.

We now describe a simulation experiment which shows the severe degradation caused by wild oscillation. Figure 4.3 shows the topology of a three-node network and the corresponding link capacity. The simulator uses TCP with show-start and congestion avoidance [Jaco88] and the SPF routing algorithm which measures the link delay every 1 second and updates the routing table if the change of delay exceeds 40 ms. Initially there is no traffic in the network and then node A initiates two TCP connections to node B and C in an arbitrary order.



Route Updating Period: 1 second

Delay Updating Threshold: 40 ms

Maximum Queue Length: 10

Average Packet Length: 512 Bytes

TCP Connections: A-B and A-C

Figure 4.3: Simulation Topology

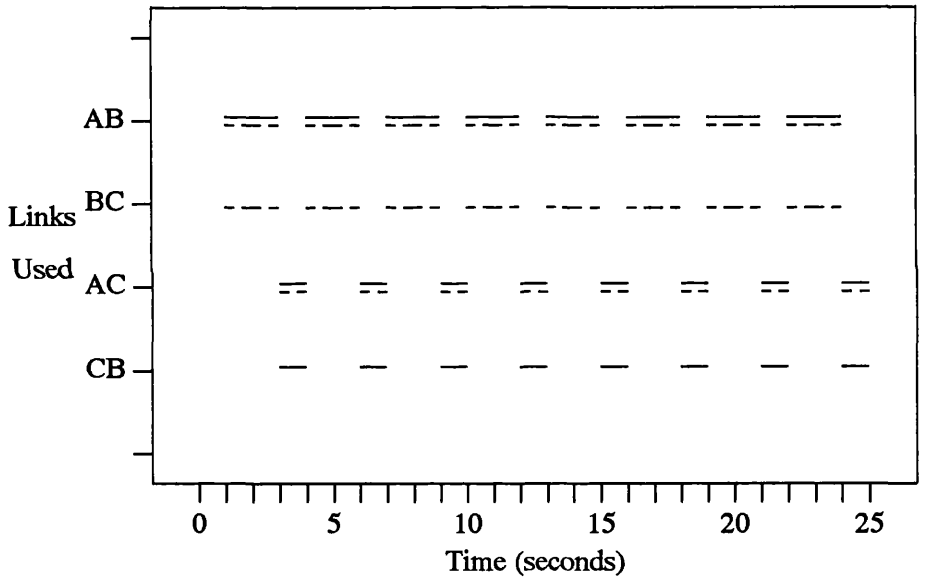


Figure 4.4: Shortest-Paths Used by Two Connections

The links used by the shortest-paths for 25 route updating periods are recorded in Figure 4.4. The solid lines and dashed lines indicate respectively the shortest-paths for connection A-B and A-C. Clearly no equilibrium is achieved and the traffic of both connections oscillates between two paths. In fact, at any given time, one of the two links AB and AC is used by both connections and the other one is left idle. This is caused by the simple decision making based on individual rationality. Each user always attempts to choose the best path. As a result, the two users always select the same path.

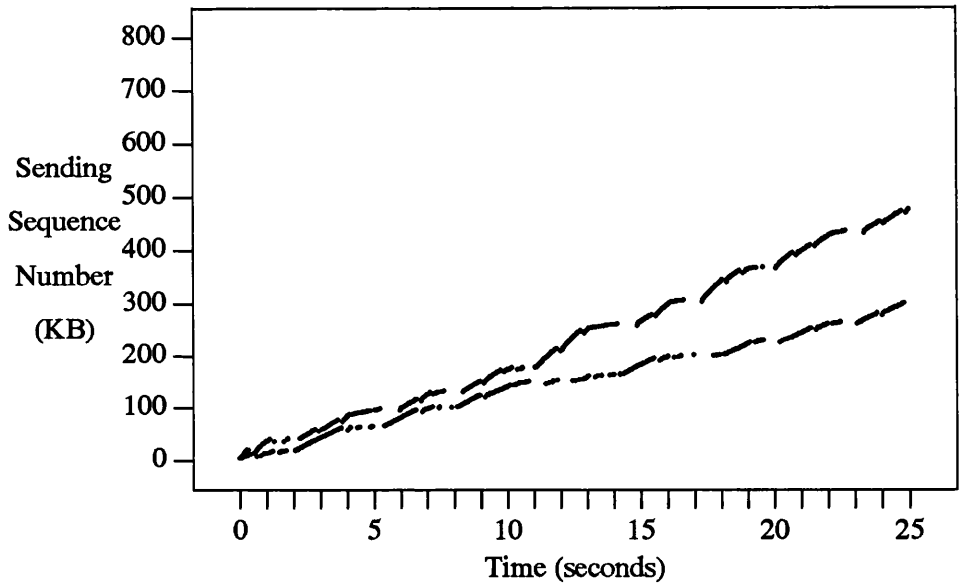


Figure 4.5: Sending Sequence Number with SPF Routing

Figure 4.5 shows the sending sequence number of the two connections with the SPF routing algorithm. The oscillation has severely affected the throughput. The total average throughput of the two connections during the 25 route updating periods is about 240 kbps which only accounts for less than half of the total bandwidth of the two links (500 kbps).

When the SPF routing algorithm can not converge to an equilibrium, its performance is in fact worse than that of a simple fixed routing scheme. Figure 4.6 shows the result of the experiment with a fixed routing scheme in which connection A-B and A-C use fixed paths link AB and link AC respectively. The total average throughput is about 450 kbps.

In our experiment, we use a sufficiently large receiving window to allow traffic load approaching the link capacity. However, due to the effect of the slow-start and congestion control in TCP source, no severe congestion occurs in the simulation experiments. The loss of bandwidth is not because of the retransmission but a result of poor routing decisions. Without congestion control in the TCP source, the performance of the SPF routing algorithm can be much worse as congestion caused by the poor routing decisions only further decreases the throughput.

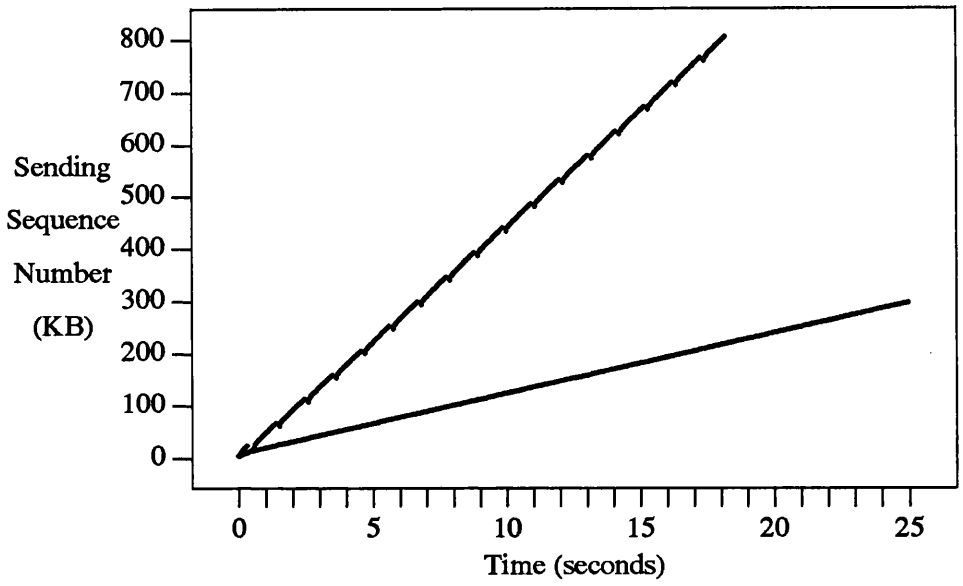


Figure 4.6: Sending Sequence Number with Fixed Routing

4.5. Responsiveness

Shortest-path routing algorithms are designed to adapt to changing conditions. It is desirable that they respond to any traffic changes dynamically. However, in a large network with heavy traffic, high responsiveness can be difficult to achieve [Seeg86].

Route updating is a costly operation in terms of CPU and bandwidth resources. Particularly for the link-state algorithms, information about the traffic changes has to be propagated to all the nodes in the network and routing tables have to be re-computed. Frequent route updating may consume a substantial amount of link bandwidth and CPU resource.

Route updating is often too slow as compared with the traffic changes, as route updating involves propagation of information across the network. The propagation delay poses a limit on how fast the routing algorithm can react to the traffic changes. In a large network, it may take a long time for routing information to travel across the network. The routing algorithm can not respond to the network conditions at a rate faster than the rate at which relevant information can reach the points concerned and corresponding action can be taken.

During the route updating, the network is in a state of transition, which may temporarily disturb the normal operation. The routing tables among the nodes are inconsistent and temporary routing loops may be formed. Moreover, the routing updates have higher priority than users' traffic. Transmission of large amount of such packets can affect the flow of the users' traffic. Hence, routing updating in a large network is usually too costly and slow for adapting to traffic

changes. Frequent route updating, under heavy traffic, can lead to wild oscillation and degradation of performance. In practical, routing updates are generated only when the traffic changes are long lasting.

4.6. Maximum Flow

Maximum flow is one of the important characteristics of a network. It represents the maximum traffic rate that the network is able to support for one connection. When the traffic is bursty, the momentary traffic rate can be many times higher than the average rate. Some time-critical applications may not allow their traffic to be flow controlled at source. It is highly desirable that the network be able to handle traffic bursts of high-speed.

According to the well-known *Max-Flow Min-Cut Theorem*, the maximum flow between any two arbitrary nodes in a network is equal to the capacity of the minimum cut separating those two nodes. However, the shortest-path routing algorithms can usually achieve far less than the theoretic potential. This problem is inherent in the single-path restriction. Since the nodes can not route traffic simultaneously to more than one path, the maximum throughput for one connection can not exceed the capacity of the best path unless the routing can change so frequently that more than one path is kept busy at one time. Consider that node x has two paths P_1 and P_2 to destination z. If at the time when the shortest-path changes from P_1 to P_2, the output queue for P_1 is full and the shortest-path changes back to P_1 before the queue for P_1 drops to zero, node x may keep both paths busy all the time and therefore increase the throughput. Suppose that Q_m, P_s and C are respectively the maximum output queue length, the average packet size and capacity of the paths. To keep n paths busy, the routing algorithm has to select each path at least once before Q_m packets drain out in $\frac{Q_m \times P_s}{C}$ time. Therefore, the maximum route updating period is $\frac{Q_m \times P_s}{C \times (n-1)}$. The maximum route updating period is often too small even in low-speed networks. Take the ARPANET for example (Q_m = 8, P_s = 512 Byte, C = 50 kbps), to keep two output lines busy, the node has to at least update routing tables every 655 ms. In high-speed networks, the figure is much smaller and impractical.

In fact, the stability problems eliminate the possibility of using route updating as means of load sharing. In reality, the route updating period is usually set quite long (eg. 10 seconds in ARPANET) to ensure the stability of the network. The maximum flow is usually around the capacity of the best path.

4.7. Congestion Control

In principle, routing algorithms with traffic-sensitive metrics have the ability to control congestion. When a path is overloaded, the reported link cost increases. The routing algorithms re-computes the routing table and reduces the traffic over the congested path.

Nevertheless, this ability is rather limited. The amount of traffic that is shed from the congested link is difficult to predict and largely depends on the composition of the traffic flow. When the traffic consists of many small flows over different source-destination pairs, appropriate amount of traffic may be shed by carefully tuning the link metrics. But if traffic in the network is dominated by large flows, the estimation errors can be very large. The routing algorithm tends to shift traffic around. It often can not converge to a new equilibrium but results in oscillation.

Congestion occurs when the traffic and resource mismatch at some points of the network. It is therefore usually local and changes rapidly. Changing routing tables, which involves exchanges of information and computation across the entire network, is often too costly and too slow. When congestion does occur, the routing algorithm has to wait until next updating period to respond. At that time, the congestion may have already dissipated. And the reported high link cost caused by the congestion, which is no longer valid, has a misleading effect on the traffic that shares the same link when the next route updating is due.

4.8. Failure Transparency

Shortest-path routing algorithms provide a certain degree of failure transparency to the network users. When resource failures are detected, the routing algorithms are able to search for alternative paths and update the routing table. For some real-time applications, it is important that the alternative paths are found quickly so that the transition can be smooth and applications do not perceive the changes.

However, it is usually difficult to detect the failures within a short time after they occur. Nodes usually rely on timeout mechanisms or reports from network management systems to monitor the availability of the resources. It can take several seconds or even minutes before a failure is detected. The traffic may still be routed along the failed path until the next routing updating. During this period, some real-time applications can be severely affected and some connections may not survive at all. The packets which are sent before the alternative paths are found may back up in the network and cause congestion and affect other traffic flows.

4.9. Summary

The behavior of shortest-path routing algorithms is examined. It has been shown that in a dynamic environment shortest-path routing algorithms often exhibit instability, produce poor routes and consequently degrade the network performance. When the traffic is heavy and changing rapidly, the estimation errors of the link distances increase and the routing algorithm often can not converge to a stable solution and instead oscillates wildly.

CHAPTER FIVE

A DYNAMIC ROUTING ALGORITHM

This chapter introduces a new approach for dynamic routing based on the concept of decentralized decision making. A new routing algorithm based on this approach, *Shortest Path First with Emergency Exits (SPF-EE)*, is presented. Extensive simulation shows that the SPF-EE algorithm improves the performance of routing in a dynamic environment.

5.1. Introduction

As has been discussed in chapter three, the route computation procedures in distance-vector and link-state routing algorithms are different. In distance-vector algorithms, the computation of routes is distributed over the network and each node accomplishes its part in the computation based on the local link distance information. In link-state algorithms, the information is disseminated to all nodes and each node calculates the routes independently. Nevertheless, a fundamental point that both distance-vector and link-state algorithms have in common is that the decision making is centralized. Although the route computation in distance-vector and link-state algorithms are carried out in different ways, the final result, a set of shortest paths for any pair of nodes or the global routing table, is in fact identical since the same link distance information is used for the computation of the shortest paths. The packet forwarding tables in different nodes may appear to be different but they are in fact derived from the same global routing table, and are therefore always consistent. This feature ensures the routing is always loop-free when the route computation terminates.

The problem of such an approach is that any route updating has to be carry out globally whenever a change in the link distances is detected. Due to the delay and the cost involved in information propagation and route computation, such global route updating is effective and worthwhile only when the change in the link distances is long-lasting. Any attempts to disperse

temporary congestion with global routing update only lead to chaos.

In this chapter we present a new approach based on the concept of decentralized decision making. In our approach, each node normally forwards packets along the shortest paths derived from the global routing table. Nevertheless, when a link failure or severe congestion is detected on the shortest path, a node is allowed to make local updating and alter its forwarding table without initiating a global route updating. Global updating is carried out only when the changes are persistent. The idea is to provide almost instant bypasses when the shortest path is in trouble and to achieve the full potential of a best effort delivery system such as a datagram network.

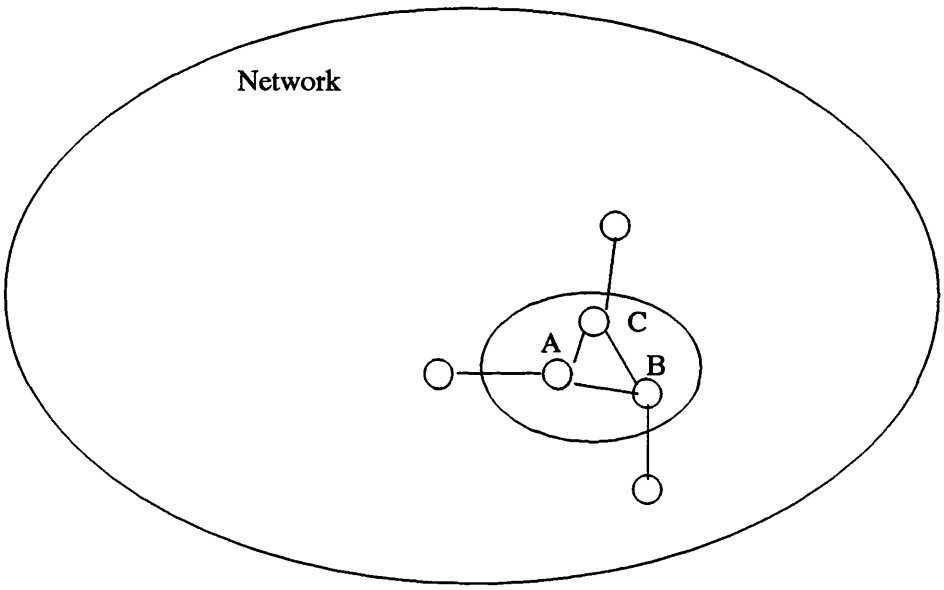


Figure 5.1: A Large Network

Let us now look at an example. Figure 5.1 shows a network, where A, B and C are three of many nodes in the network. Suppose that at time t node A detects an increase in the distance of link AB. Instead of initiating a global route update, node A may consult its neighbour nodes and find a temporary bypass to the destination which avoids link AB. Such a bypass can be found by limited message-passing and computation within the neighbourhood of node A. If the problem with link AB is persistent, the increase in the distance of link AB will be eventually reflected in the global routing table when the route updating time is due.

In the rest of this chapter we present a new routing algorithm based the principles of decentralized decision making, and discuss its performance.

5.2. Overview

The new routing algorithm can be viewed as an extension to the SPF algorithm. The strategy is to maintain a stable global routing table and meanwhile provide mechanisms to disperse traffic temporarily when it is accumulating in some particular areas of the network. When congestion and network failures do occur, traffic is forwarded along the alternative paths temporarily and bypasses the congested or failed areas. The new algorithm carries out all the functions that the SPF algorithm does. In addition to that, the new algorithm can make local routing update and provide *emergency exits* for the traffic when the shortest paths are experiencing problems. Because of this feature, we call the new routing algorithm *Shortest Path First with Emergency Exits (SPF-EE).*

As emergency exits, the alternative paths are not meant to be the shortest paths to the particular destination. The alternative paths and the shortest path may not be disjoint. The aim of the alternative paths is to provide an almost instant bypass for the traffic to a congested or failed area. Such adjustment of routing is temporary and only involves neighbour nodes nearby.

Under the SPF-EE algorithm, a node x normally forwards a packet for destination z to neighbor NS_z, where NS_z is the next-hop along the shortest path (SP) to the destination z. But if the queue length to node NS_z (denoted by Q_{NS_z}) exceeds a certain limit, the packet is transmitted to neighbor NA_z, where NA_z is the next-hop along the alternative path (AP).

When a packet is forwarded to a neighbor y other than the next-hop along the SP, there are only two possibilities. If the neighbor y is not upstream from the node x in the sink tree for the destination z, node x can send the packet for destination z to node y and they will travel along a SP from node y to the destination z (Figure 5.2(a)) without forming any loops. We call node y an *exit* for the destination z.

However, if all the neighbors other than node NS_z are upstream from node x, the packet will be looped back to node x. In this case, node x sends a control packet to all its neighbors other than NS_z to inquire whether they have any exits for the destination z. Upon receiving the control packet, each neighbor checks to see whether it has a neighbor which is not upstream node from node x. If an exit is found, it sends a reply back and establishes a *reverse alternative path* (RAP). Otherwise, it propagates the control packet further to its neighbors until an exit is found. When using a RAP, node i has to source-route the packet to the exit (Figure 5.2(b)). In some cases, the packet may be sent backwards to the node it just comes from. AP can be viewed as a special case of RAP. With APs the exits are next-hop while with RAPs the exits are

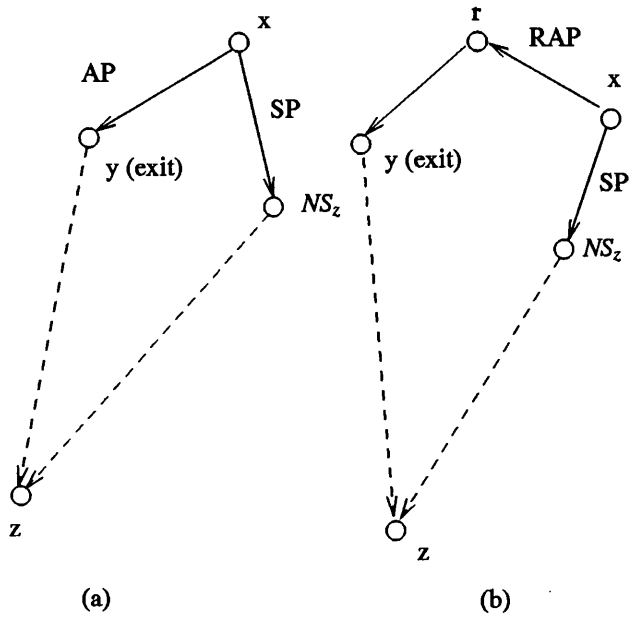


Figure 5.2: Alternative Path and Reverse Alternative Path

more than one hop away.

The algorithm fails when none of the upstream nodes have contact with other branches of the sink tree. This occurs only when the network is partitioned.

To illustrate the algorithm, consider a six-node network of Figure 5.3(a), with the routing tree for node A shown in Figure 5.3(b).

Traffic destined to node C is normally routed along the shortest path via node B. However, if for some reason the link AB is congested, node A has to find an alternative path to node C. According to the sink tree for node C shown in Figure 5.4(a), node A has a neighbor node D which is not upstream from node A in the sink tree for node C. Thus an AP is available via node D.

For destination F the situation is different as both neighbors (B and D) are upstream from A (Figure 5.4(b)). Consequently, node A sends each of them a control packet. Node D replies that it has an exit node E. Node A therefore records the RAP and source-routes the packet along the route A-D-E.

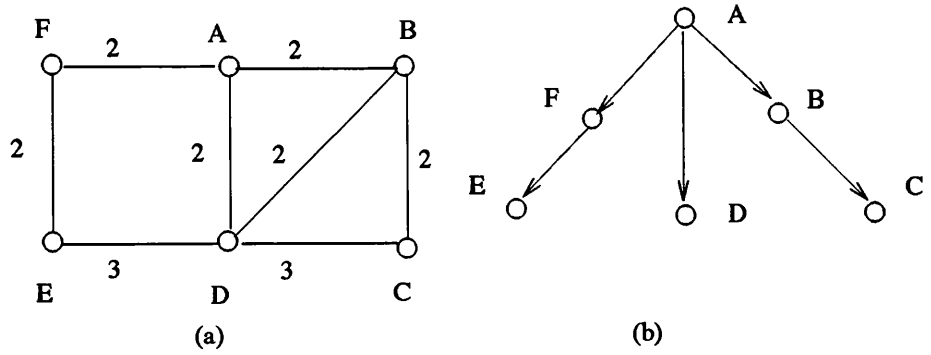


Figure 5.3: Network Topology and Routing Tree for Node A

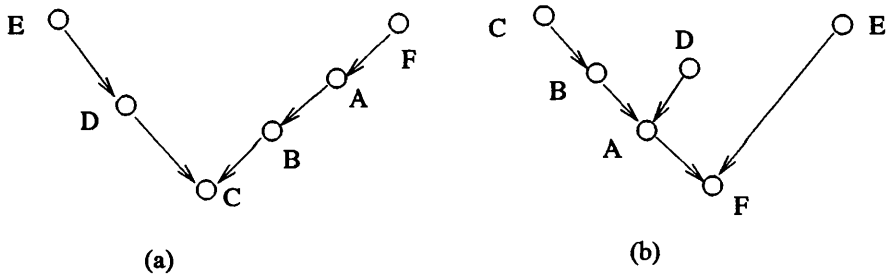


Figure 5.4: Sink Trees for Node C and Node F

5.3. Description of the Algorithm

We now describe the route computation and packet forwarding procedures in some detail. A more formal version of the entire algorithm is presented in Appendix.

5.3.1. Route Computation

In the SPF-EE algorithm, each node maintains a routing table with one row per destination, as in Table 5.1(a). The first column gives the next-hop along SP. The second column is dedicated to the APs. If an AP is not available, it is left blank. In addition to this, there is a RAP table for maintaining the the RAPs (Table 5.1(b)).

The APs can be easily derived from the topological database within the node. They are calculated at the time of route updating along with the SPs. The calculation of RAP involves communications with other nodes. They are generated on an event-driven basis.

In theory, the maximum number of possible APs equals to the number of neighbors other than the one along the shortest-path. However, as emergency exits, they are much less frequently used than the SPs. Thus only one AP per destination are calculated at the time of route updating. Nevertheless, any AP or RAP can be calculated dynamically on request.

When a node x receives a routing update, it first calculates the SPs for all the destinations, which is the same as the SPF algorithm. The result is a routing tree rooted at node x (denoted by $TREE_x$) and the first column of routing table for the next-hops of the SPs. It then derives the routing trees for each of its neighbors $y \in N_x$, where N_x is the set of neighbors of node x.

When the routing tree for node x is known, the calculation of the routing tree for an adjacent node y only requires only an incremental calculation rather than a complete re-calculation [McQu80]. When the routing tree for node x is converted to the routing tree for node y, any nodes in the subtree rooted at node y will not be repositioned as these nodes arealready at minimum delay. Therefore only nodes that are not on the subtree rooted at node y need to examine their neighbors to see if there is a shorter path.

For the routing tree $TREE_y$, $y \in N_x$, if a destination node z is not on the subtree rooted at node x, and node y is not the next-hop of the SP for it, node y is the exit for destination z.

If for a destination node z, no APs can be not found, a RAP will be established when the length of outgoing queue to destination z exceeds a threshold. To establish a RAP, node x sends a *query* message (denoted by MSG(query,x,z)) to its neighbors y ($y \in N_x$, $y \neq NS_z$). The message contains a message ID, addresses of node x and r. Node x then marks the entry for destination z in the RAP table as

waiting , which prevents further query messages being sent before it receives a reply. Upon receiving MSG(query,x,z), node r checks in the routing tree $TREE_m$ ($m \in N_r$, $m \neq x$) whether destination z is on the subtrees looted at node x and r. If it is not, node m can be the exit for destination z.

Node r records the addresses of node x, z and m in the RAP table, includes the addresses of node r and m into a *reply* message (denoted by MSG(reply,x,z)) and sends it back to node x. Otherwise if no exits are found, node r includes its address and continues to propagate to its neighbors. If an exit is not found within a number of hops, the search is given up as a better

	SP	RAP
A	-	-
B	B	D
C	B	D
D	D	B
E	F	D
F	F	-

(a)

Source	Destination	Intermediate Hops	Exit
A	F	D	E

(b)

Table 5.1: Routing Table for Node A

exit is likely to be found along other paths. If no RAPs can be found at all, the entry marked busy will prevent further query messages being sent until a routing update is received and the RAP table is reset. This is because RAPs may not exist in some topologies.

When node x receives a reply, it records the route to the exit in the RAP routing table. Node x may receive more than one reply message for a query message, it only records the first one it receives and ignores the rest.

If node r receives a routing update, it checks the RAP table. If the source of one entry is different from the current node, it sends a *reset* message (denoted by MSG(reset,x,z)) to that source and clears that entry. When node x receives a reset message from node r, it clears the entry in the RAP table that matches node x, z, r and m. If node x receives a routing update, it clears the RAP table. The reset message is used to ensure that RAPs which are established

when the databases in node x and r are not consistent will be reset when they become consistent, and therefore to prevent long-lasting loops in RAPs.

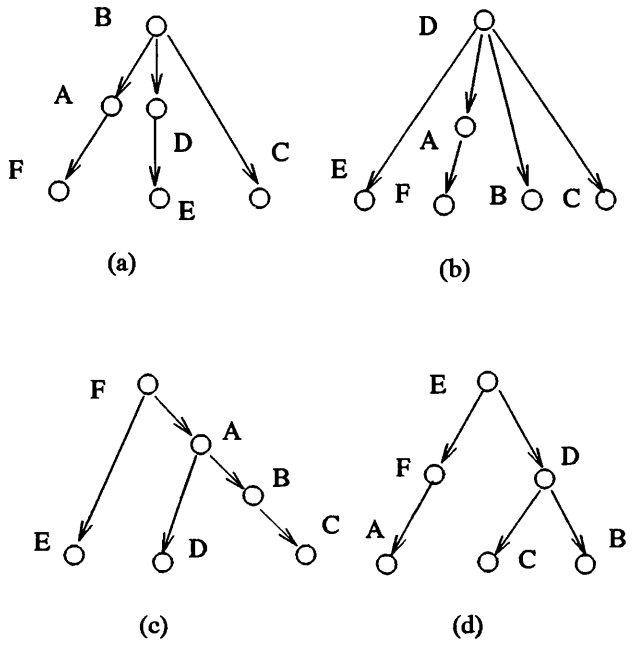


Figure 5.5: Routing Trees for Node B, D, F and E

Figure 5.5 shows the routing trees for node B, D, F and E. Take routing tree for node B for example, node D and E are not on the subtree rooted at node A, and node B is not their next-hop for the SP. Therefore node B is the next-hop of the APs for destination D and E. After all APs are filled into the routing table (Table 5.1(a)), only destination F does not have an AP entry. When the link AF is congested, node A sends a query message to node D and B. After checking the routing tree for node E, node D finds an exit node E and replies back to node A. Node A records a RAP A-D-E for destination F.

5.3.2. Packet Forwarding

The packet forwarding in the SPF algorithm is simple: a node transmits the packets to next-hop of the SP as indicated by the routing table. However, in the SPF-EE algorithm, the choice of the next-hop depends on the length of the outgoing queues. There are two predetermined parameters used in the SPF-EE algorithm for forwarding: T_s (threshold of queue length for

node NS_z) and T_a (threshold of queue length for node NA_z). Before a node forwards a packet to the next-hop, it checks the outgoing queue. If the queue length to the next-hop exceeds the threshold, it attempts to avoid using that link.

Figure 5.6 shows a flowchart of the forwarding algorithm. When a node receives a packet, it first attempts to forward along the SP. If the node decides that the SP is congested, it turns to the AP or RAP. If the AP or RAP are not available or they are also congested, the node starts to search for a new AP or RAP meanwhile it queues the packet along the SP.

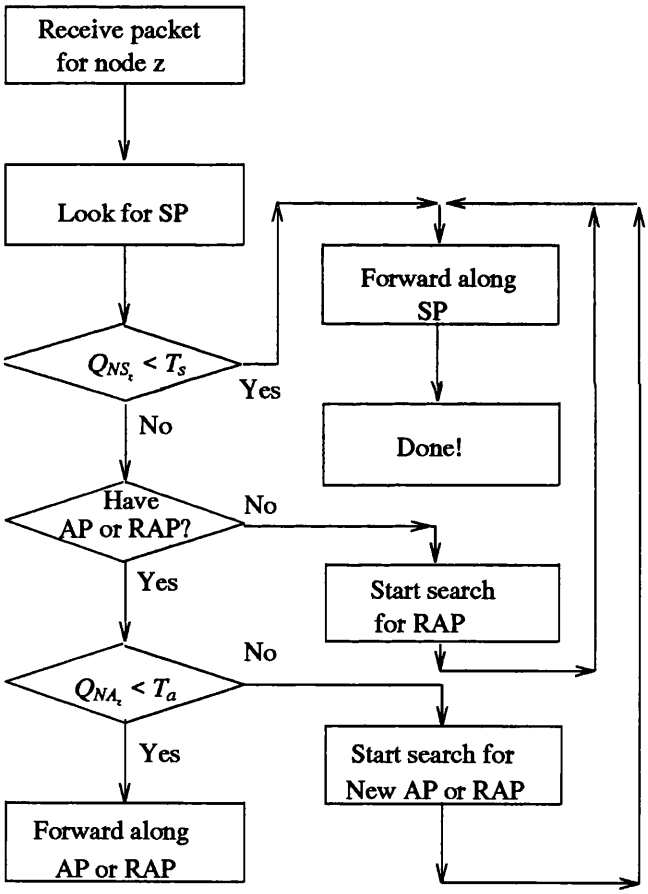
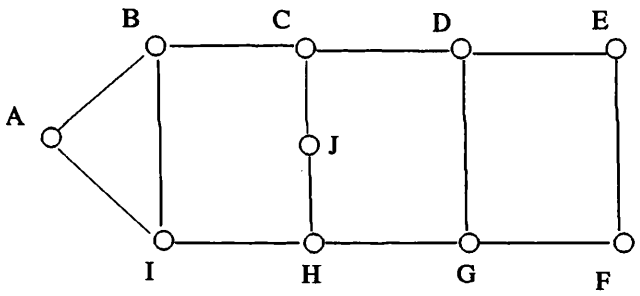


Figure 5.6: Forwarding Algorithm

5.4. Performance Comparison

In this section, we present some simulation results on the performance of the SPF-EE algorithm. Since the SPF-EE algorithm is an attempt to eliminate certain problems with the SPF algorithm under heavy and dynamic traffic, the intention of the simulation is to compare the performance of the SPF-EE algorithm and the SPF algorithm, and demonstrate the improvement of the SPF-EE algorithm over the SPF algorithm with regards to the problems discussed in chapter four.

Figure 5.7 shows the topology and some environment parameters used in the simulation. Experiments use both the ill-behaved TCP sources (TCP without congestion control) and the well-behaved TCP sources (TCP with slow-start, congestion avoidance and exponential retransmission backoff [Jaco88]). The simulator is a modified version of the MIT Network Simulator [Heyb89].



Link Speed: 200 KB/sec, Propagation Delay: 50 usec

Node Processing Delay: 10 usec, Maximum Queue Length: 10

T_a and T_s: 5, RAP Cache Timeout: 5 seconds

Figure 5.7: Simulation Topology and Environment Parameters

5.4.1. Maximum Flow

With the SPF algorithm the path for a pair of source and destination is calculated at the time of route updating and remains unchanged throughout one route updating period (eg. 10s in SPF), while with the SPF-EE the path is not predetermined, the actual route is decided by individual node according to the real-time traffic conditions and more than one paths can be open to a

connection. In the sense, it has the ability of loadsharing over different paths.

Figure 5.8 shows the sending sequence number of a well-behaved TCP connection between node C and H with an excessive window size (15360KB).

With the SPF algorithm, the SP between node C and H oscillates among CBIH, CJH and CDGH and the throughput saturates around 20 KB/sec. With the SPF-EE algorithm, however, after some initial stage, all the three paths (CBIH, CJH and CDGH) are open for this connection and the throughput approaches 60 KB/sec, which is the theoretic limit between the two nodes. The gaps in the curve indicate packet drops and retransmits.

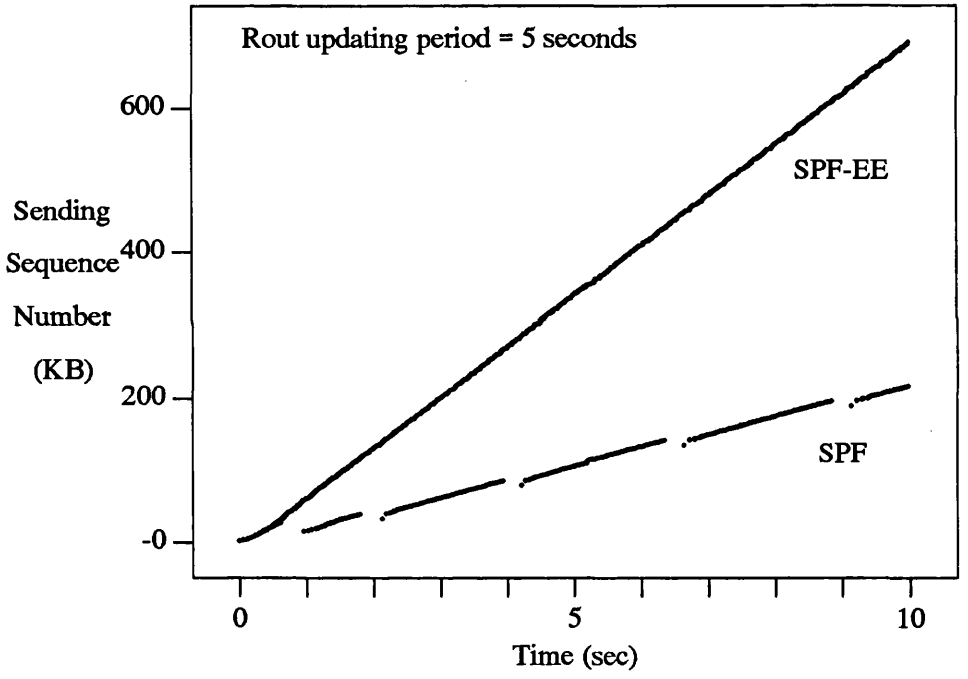


Figure 5.8: Sending Sequence Number of a well-behaved TCP Connection

Due to the effect of the congestion control in the TCP sources, the paths between node C and H are not severely congested. Figure 5.9 shows the sending sequence number of a ill-behaved TCP connection. The SPF algorithm performs much worse without congestion control in the TCP source. Congestion and retransmission severely affects the throughput. The curve for the SPF-EE algorithm in Figure 5.9 is almost the same as the one in Figure 5.8 except the sharp rise at the beginning in Figure 5.9, which is due to the fact the ill-behaved TCP source does not have the slow-start mechanism.

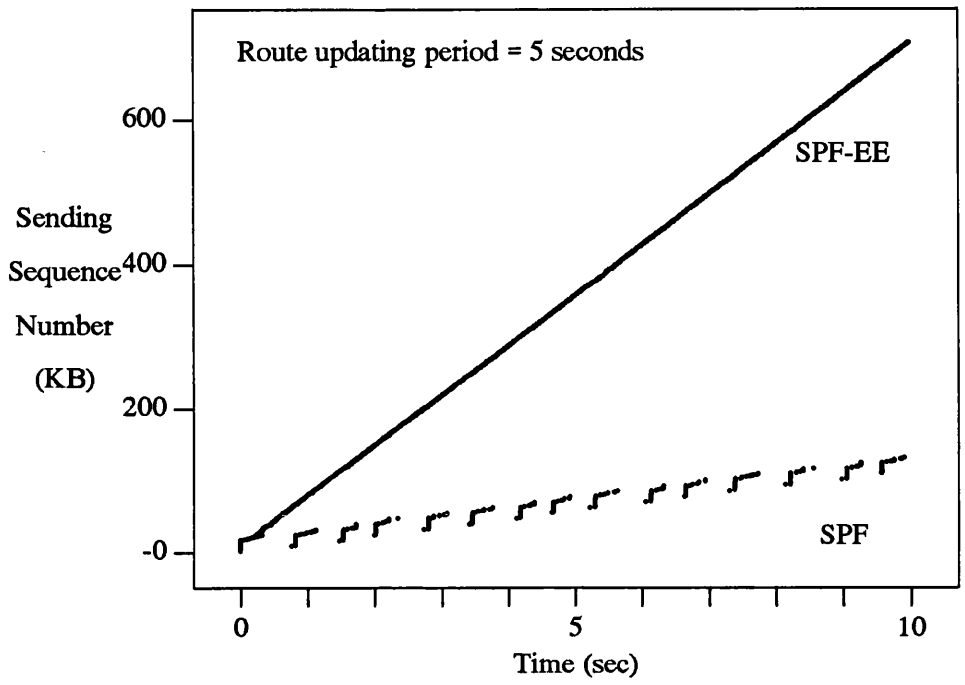


Figure 5.9: Sending Sequence Number of an ill-behaved TCP Connection

5.4.2. Oscillation

The SPF-EE algorithm can not eliminate the oscillation completely. In fact, with delay-based link metrics, oscillation is inevitable under heavy traffic load. Nevertheless, the SPF-EE algorithm can drastically reduce the negative effects of oscillation. When links are congested, the queueing delay increases significantly. At next route updating, it is likely that the reported the delay of the links will be so high that those links will be excluded from all routing trees of any sources, as a result, those links will be idle for one entire route updating period while some others may be severely congested. Moreover, those links will report low delays when next route updating is due and become congested again.

With the SPF-EE algorithm, when the outgoing queue of a node exceeds the threshold, the node opens a new channel for the accumulated packets, which effectively reduces the congestion and queueing delay. Thus, the reported link delay does not oscillate so dramatically. Even if some links do become unattractive to all sources as a result of high reported delays, they will still be used as APs. As in Figure 5.8, although the SP between node C and H still oscillates between CBIH, CJH and CDGH, three paths are all in use.

5.4.3. Overhead and Responsiveness

The SPF-EE algorithm needs more storage and CPU resources in the node than the SPF algorithm. The routing table in the SPF-EE algorithm is slightly larger than the one in the SPF algorithm as there is one column for the APs and additional table for RAPs. However, the SPF-EE algorithm uses the same topology table as the SPF algorithm does, therefore the increase in the storage is limited. For each route updating, the SPF-EE algorithm calculates routing trees for itself and its neighbors, thus it needs approximately D times computation as the SPF algorithm needs, where D is the average degree of the nodes. The messages for establishing the RAPs are rather small and the information exchanges are confined in local areas. Once a RAP is established, it is likely to be used until next route updating point. The entry is kept in the cache until timeouts. Therefore, the additional overhead is minor as compared with the broadcasting of route updating.

On the other hand, however, with the SPF-EE algorithm the frequency of route updating can be greatly reduced, which will significantly decrease the computation and communication overhead. With dynamically changing traffic, it is not possible to obtain an accurate estimate of average delay within very short time. In the SPF-EE algorithm, the momentary fluctuation of the traffic is handled by APs and RAPs. Route re-computation is only needed for the long-lasting topological changes (eg. resource failures). Much artificial updating in the SPF algorithm caused by dynamic traffic can be eliminated.

With the SPF-EE algorithm, the increase in the route updating period in no way reduces the responsiveness. In fact, in the SPF-EE algorithm the nodes respond to the traffic on a packet-by-packet basis and are able to find alternative paths when necessary. The dilemma of overhead and responsiveness is solved by updating the routing table to adapt the topological changes and providing alternative paths to cope with the fluctuations of traffic.

5.4.4. Congestion Control and Fault Tolerance

Most congestion control mechanisms attempt to prevent the excessive traffic from entering the network. The SPF-EE algorithm, however, achieves the congestion control by routing the traffic around the congested area. Congestion occurs often because the unevenly distributed traffic mismatches the resources available. In a large network, while a few links are congested, other links nearby may have little traffic. With the SPF algorithm, it is impossible to switch to other links without updating the routing tables over the entire network. But with the SPF-EE

algorithm, a node can react to congestion on a packet-by-packet basis. It can switch to an AP when congestion occurs and switch back when the congestion is gone. In the face of link or node failures, the SPF-EE algorithm can route the traffic along the AP until next route updating is due, which provides great transparency to the network users.

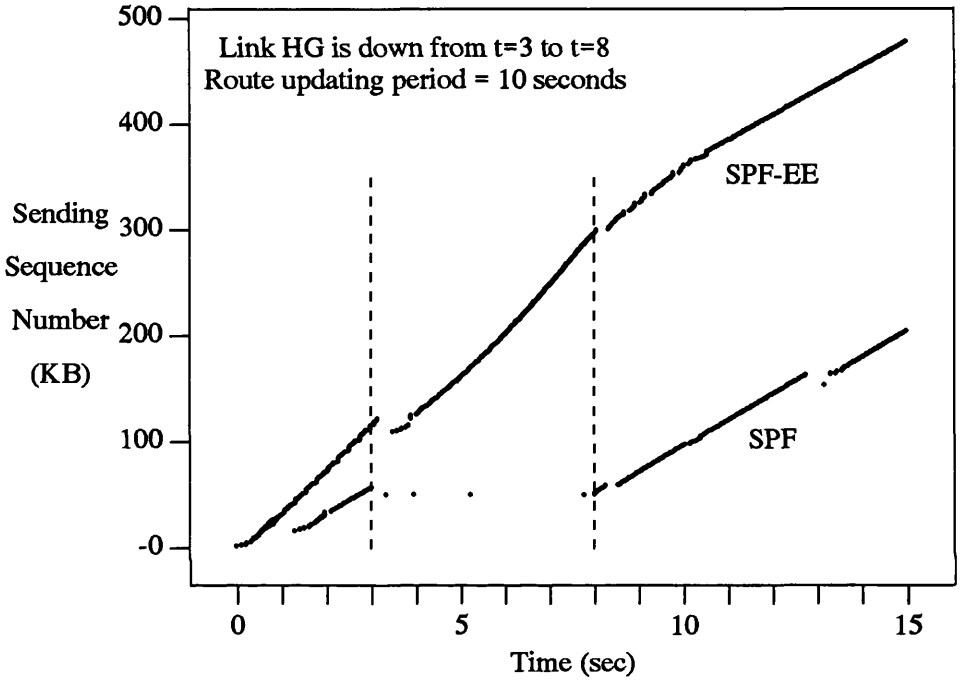


Figure 5.10: Fault Tolerance

Figure 5.10 shows the result of an experiment with link HG is down for 5 seconds. A well-behaved TCP connection is established between node A and node F in Figure 5.7. About 3 seconds after route updating when the routing table indicates that path AIHGF is used, link HG goes down and it comes up again another 5 seconds later. Figure 5.10 shows that with the SPF algorithm the traffic flow is completely prevented during the downtime. While with the SPF-EE algorithm, the connection recovers very quickly and the traffic is routed along the alternative path HJCDGF. As a matter of fact, if the packets are accumulating along the AIHGF, node A may simply forward via node B.

5.5. Summary

A new approach for dynamic routing based on the concept of decentralized decision making is introduced. A dynamic routing algorithm based on the new approach (SPF-EE) is presented, which eliminates many of the problems that the SPF algorithm exhibits under heavy and dynamic traffic. In the SPF-EE algorithm, traffic on overloaded paths can be dispersed to alternative paths without route updating and bypass areas where resources are scarce. The SPF-EE algorithm tends to equalize the traffic distribution over the entire network therefore it can utilize the resources more efficient. Simulation has shown that the SPF-EE algorithm has achieved substantial improvement over the SPF algorithm.

CHAPTER SIX

INFORMATION FEEDBACK IN CONGESTION CONTROL

This chapter examines the information feedback mechanisms in congestion control. Two new binary feedback schemes and one new quantitative feedback scheme are proposed.

6.1. Introduction

In a high speed network, traffic can be generally divided into two categories: real-time and non-real-time. Real-time traffic such as voice and video has stringent delay requirements therefore can not be easily flow-controlled. Reservation or prioritised access are often given to real-time traffic to ensure rigid performance guarantees. Non-real-time traffic such as file transfers, on the other hand, has to use the remaining network capacity and is subject to congestion control. The sender of non-real-time traffic is required to adjust its traffic rate or window size based on the feedback information collected by intermediate routers explicitly [Rama88] or derived from the acknowledgements implicitly [Jain86, Jain89, Jaco88].

There are two basic approaches for information feedback: *binary approach* and *quantitative approach*. In the binary approach, the monitor component sends a signal to the source whenever it detects congestion. The feedback information is *binary*, in the sense that it only provides a direction of the adjustment ie. increase or decrease. To approach the operating point, recursive increase and decrease operations are required. In the quantitative approach, the monitor component attempts to estimate the current traffic load and provide more quantitative information which may enable the source to adjust the traffic more accurately and with much fewer operations.

In this chapter we first examine several existing information feedback mechanisms, then present two new binary schemes and one new quantitative scheme. These mechanisms described

here will be used, in conjunction with a new traffic adjustment algorithm, to form complete congestion control schemes in the next chapter.

6.2. Binary Feedback

Three different binary schemes have been proposed for congestion signalling. In the DEC-BIT scheme [Rama88], the queue length at the switches is monitored. When the queue length exceeds one, the signal is fed back by setting the congestion indicator bit in the packet headers. This approach needs one bit in the packet header and the switches have to re-compute the check-sum if they have set the congestion indicator bit. In [Jain89], Jain proposed to use *normalized delay gradient (NDG)* as the signal:

$$NDG = \frac{\delta D/\delta W}{D/W}$$

where D and W are delay and window size respectively.

the window size W is increased when $NDG \leq 0$ and decreased otherwise. Because NDG depends on the relative values between two delay measurements, it is often too sensitive to second order delay fluctuations. Timeouts on receiving acknowledgements for transmitted packets are also used as a congestion signal in [Jain86] and [Jaco88]. Since the transmission error rate is usually very low, timeout is most likely caused by packet losses as a result of buffer overflow. The drawbacks of using timeout as a congestion signal is that the warning comes quite late and packet losses are inevitable.

We now present two new binary feeedback schemes based on throughput gradient and delay threshold.

6.2.1. Throughput Gradient Scheme

The throughput gradient scheme is based on continuous evaluation of the current throughput gradient. Consider the general characteristic of the network total throughput as a function of offered load, as illustrated in Figure 6.1. Throughput increases linearly with the traffic load under light traffic and levels off when the path is saturated. The traffic load at the turning point also rises when the resources at the bottleneck increases. The gradient of the throughput curve can be used as the indicator of the resource utilization on that path. We define

$$\textit{Throughput Gradient } TG(W_n) = \frac{T(W_n)-T(W_{n-1})}{W_n-W_{n-1}}$$

where W_n and W_{n-1} represent the two sequential window sizes and $T(W_n)$ is the throughput at window size of W_n. Although *TG* decreases towards zero as the traffic load increases, the actual values of the *TG* that the users have are different as they depend on round trip delays. The actual metric used in our scheme is called *normalized throughput gradient (NTG)*, which is defined as

$$NTG(W_n) = \frac{TG(W_n)}{TG(W_1)}$$

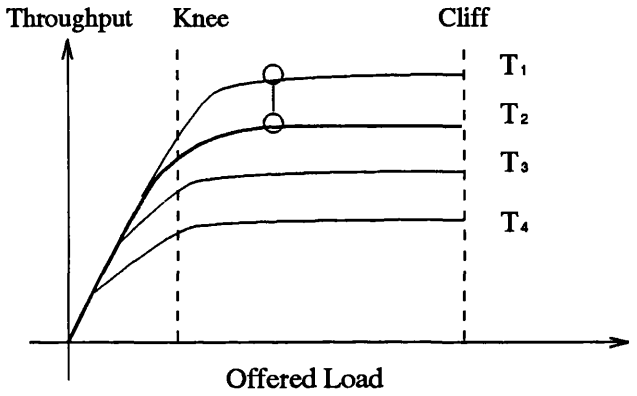


Figure 6.1: Total Network Throughput as a Function of Offered Load

As the traffic load changes, the *NTG* varies approximately in the range [1,0]. Under light traffic, the *NTG* is around 1 since the increase in throughput is approximately proportional to the increase in traffic load. The *NTG* decreases gradually as the traffic load increases and reaches around zero when the path is saturated.

When resources are released, the *NTG* may rise substantially since for each user it is equivalent to the resources at the bottleneck being increased. As can be seen in Figure 6.1, the throughput can jump from throughput T_2 to a higher level T_1 when the resources are increased.

6.2.2. Delay Threshold Scheme

The delay threshold scheme is based on the fact that the round trip delay has two major components: the round trip propagation delay and the overall queueing delay. Therefore, by regulating the queueing delay, the network can be maintained at a proper operating point.

Suppose that D_p and $D_q(Q_n)$ are respectively the round trip propagation delay and the overall queueing delay that the packet n has experienced, where Q_n is the overall queue length that packet n has experienced. Therefore the round trip delay that packet n has experienced $D(n) = D_p + D_q(Q_n)$. Let M and μ be the packet size and the bottleneck capacity, we have

$D_q(Q_n) = \frac{Q_n \times M}{\mu}$. Therefore, the minimum round trip delay $D_{min} = D(0) = D_p$ and the maximum round trip delay $D_{max} = D(Q_{max}) = D_p + \frac{Q_{max} \times M}{\mu}$, where Q_{max} is the maximum queue length. We assume that the retransmission timeout estimate is accurate. When packets are lost due to the overflow of buffers, $Q \approx Q_{max}$ and $D \approx D_{max}$. In the sense, timeout can be viewed as a queue length signal threshold of Q_{max}. To avoid the periodic losses, the queue length threshold has to be much lower than Q_{max}. We define the queue length signal threshold $Q_i = \alpha Q_{max}$ ($\alpha < 1$). The corresponding delay signal threshold $D_i = D_p + \alpha \frac{Q_{max} M}{\mu}$. After some manipulation, we obtain $D_i = (1-\alpha)D_{min} + \alpha D_{max}$. The parameter α determines the operating point that the congestion control algorithm attempts to maintain. Due to the delay in feedback information, the actual Q and D will oscillate around the operating point. The choice of α depends on how the operating point is defined. In our simulation in the next chapter, we choose $\alpha = 0.5$, ie. $Q_i = Q_{max}/2$, as the operating point, which allows maximum queue fluctuation both upwards and downwards. We can therefore get $D_i = (D_{min} + D_{max})/2$. In other words, we attempt to maintain RTT in the middle of its maximum and minimum range. We also approximate D_{min} and D_{max} with the current minimum and maximum of round trip delay.

6.3. Quantitative Feedback

To speed up traffic adjustment and avoid oscillation, it is desirable that quantitative information on the network state can be obtained so that the sender can adjust its traffic rate or window size quickly and accurately in one step. In recent work, Mitra and Seery [Mitr90] proposed a window adjustment scheme based on some results from asymptotically optimal design of virtual circuits. In [Kesh91a], back-to-back Packet-Pairs are used to probe the bottleneck capacity in a network with fair-queueing disciplines [Deme89] in the switches. Keshav [Kesh91c] also discussed the estimators to be used with the Packet-Pair scheme.

In this section we present a new approach for deriving quantitative information for rate-based congestion schemes. Based on the fluid model, we show that the bottleneck capacity can be estimated from round trip delay when the sender is increasing its traffic rate linearly. The exact solution for time-dependent M(t)/M(t)/1 queue and simulation results show that the approximation is accurate when the traffic load is high.

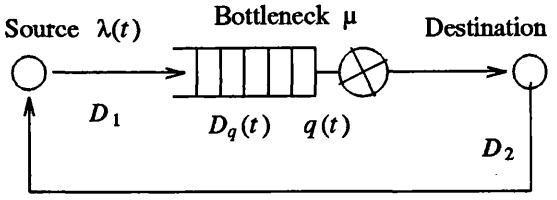


Figure 6.2: Network Model

6.4. Fluid Model Approach

Figure 6.2 shows a simple model of a network with one bottleneck. We assume that the receiver sends back an acknowledgement each time when a packet arrives. The parameters describing the model are as follows:

$\lambda(t)$: the source traffic rate at time t

μ: the bottleneck service rate

$q(t)$: the queue length at the bottleneck at time t

D_1: the propagation delay between the source and the bottleneck

D_2: the propagation delay between the bottleneck and the source via the destination

$D_q(t)$: the queueing delay at the bottleneck at time t

$RTT(t)$: the round trip time calculated at time t

The problem is to estimate the bottleneck capacity μ from the round trip delay sample $RTT(t)$ when traffic $\lambda(t)$ is increasing linearly, ie. to establish $\mu = f(\lambda(t), RTT(t))$ when $\lambda(t) = \alpha t + c$.

We solve the above problem by using a fluid model approximation. The idea is to view the network as a dynamic system and consider $\lambda(t)$, $q(t)$, $D_q(t)$ and $RTT(t)$ as real-valued continuous-space deterministic functions of time. The dynamic behavior of such a system can then be described by differential equations.

Fluid model approximation or deterministic analysis has been used for design and analysis of flow control mechanisms. In [Wacl90], Waclawsky and Agrawala presented a detailed deterministic analysis of the transit behavior of window mechanisms. Bolot and Shankar studied the AIMD algorithm with fluid model approximation [Bolo90a, Bolo90b]. The design of Packet-Pair scheme is also derived from the deterministic analysis [Sing90]. There are several reasons for using a fluid model approximation instead of a stochastic queuing model:

- Most of the traffic adjustment schemes are time-varying and state-dependent systems. The analysis of such systems are difficult, and in many cases are impossible. There are only a few simple cases of time-dependent queueing which have been solved [Taka62] and numerical solutions based on Chapman-Kolmogorov differential equations also become unmanageable for complex systems [As86].
- Even if exact solutions do exist, the complexity of such solutions may make them infeasible to be used as on-line control mechanisms. The resources available for traffic control are very limited therefore the control policy has to be simple, robust and easy to implement in hardware.
- Due to the unpredictable nature of traffic, it is difficult to make any assumptions as to the characteristics of the traffic patterns. The primary goal of the control policy is then to ensure safe and expected behavior of the system. Under such circumstances, it is reasonable to use a fluid approximation (ie. first order approximation of a G/G/1 system [Klei76]) as the basis for traffic control and deal with fluctuations as noise [Kesh91c].
- Most traffic control mechanisms come into effect only when traffic load exceeds the resources and the adjustment operates at a very short time scale (ie. one or several round trip times). This heavy load and its transient nature make fluid model approximation an attractive option for reasonable accurate estimation.

6.4.1. Analytical Results

We consider a rate-based system as shown in Figure 6.2. Suppose at time t=0, $\lambda(0) = c$ and $q(0) = 0$. After t>0, the source increases its λ linearly from initial value c at a rate α, ie. $\lambda(t) = \alpha t + c$. Note that the traffic sent at time t arrives at the bottleneck at time $t+D_1$. When $\lambda(t-D_1) > \mu$, the queue at the bottleneck starts to build up. Based on the fluid model, we have

$$\begin{cases} \dot{q}(t) = \lambda(t-D_1) - \mu & \text{if } \lambda(t-D_1) > \mu \\ q(t) = 0 & \text{if } \lambda(t-D_1) \le \mu \end{cases} \tag{6.1}$$

$$\begin{cases} q(t) = q(0) + \int_0^t \lambda(t-D_1)dt - \int_0^t \mu dt & \text{if } \lambda(t-D_1) > \mu \\ q(t) = 0 & \text{if } \lambda(t-D_1) \le \mu \end{cases} \tag{6.2}$$

Since $\lambda(t) = \alpha t + c$ and let $t_0 = \frac{\mu - c}{\alpha} + D_1$, we get:

$$q(t) = \int_{t_0}^{t} (\alpha t - \alpha D_1 + c - \mu) dt \quad (t > t_0)$$

We solve the above equation and get

$$q(t) = \frac{\alpha}{2}(t - D_1 - \frac{\mu - c}{\alpha})^2 \tag{6.3}$$

Equation (6.3) is the basis from which we derive the rest of the equations. Before proceeding further, we first discuss the possible error introduced by the approximation.

We now compare (6.2) with the exact solution of a time-dependent M(t)/M(t)/1 queue with arrival rate $\lambda(t)$ and service rate $\mu(t)$. Based on [Ride76, Saat61], we have the exact solution for $q(t)$

$$q(t) = q(0) + \int_0^t \lambda(t) dt - \int_0^t \mu(t) dt + \int_0^t \mu(t) P_o(t) dt \tag{6.4}$$

Where $P_o(t)$ is the probability that there is no packets in the queue at time t. It is usually not possible to obtain an analytic solution to the equation because of the difficulty of integrating over $P_o(t)$.

Note that there is no propagation delay between the source and the bottleneck, ie. $D_1 = 0$ and $\mu(t) = \mu$. Comparing (2) and (4), we find that the difference lies in $\mu \int_0^t P_o(t) dt$. The difference becomes negligible only when the traffic load is high so that the probability that there is no packets in the queue is approximately zero, ie. $P_o(t) = 0$.

For other classes of time-dependent queues, it is difficult to obtain a simple solution to compare with (6.3). However, fluid model approximation is reasonably accurate when the queue length is larger than 1 [Newe71].

Suppose that t' and t are respectively the time when the data is sent and the corresponding time when its acknowledgement is received. We have $RTT(t) = t - t'$ or $t' = t - RTT(t)$. The $RTT(t)$ consists of round trip propagation delay and queueing delay, ie. $RTT(t) = D_1 + D_q(t' + D_1) + D_2$. After eliminating t', we get:

$$RTT(t) = D_1 + D_2 + D_q(t - RTT(t) + D_1)$$

Since $D_q(t) = \frac{q(t)}{\mu}$, it follows:

$$RTT(t) = D_1 + D_2 + \frac{q(t - RTT(t) + D_1)}{\mu} \tag{6.5}$$

Let RTT_1 and RTT_2 be two samples of $RTT(t)$ collected at time t_1 and t_2 ($t_1, t_2 > t_0$), ie. $RTT_1 = RTT(t_1)$ and $RTT_2 = RTT(t_1)$. We then have

$$RTT_1 = D_1 + D_2 + \frac{q(t_1 - RTT_1) + D_1)}{\mu}$$

$$RTT_2 = D_1 + D_2 + \frac{q(t_2 - RTT_2) + D_1)}{\mu}$$

Now we have:

$$RTT_1 - RTT_2 = \frac{q(t_1 - RTT_1) + D_1)}{\mu} - \frac{q(t_2 - RTT_2) + D_1)}{\mu}$$

$$\mu = \frac{q(t_1 - RTT_1) + D_1) - q(t_2 - RTT_2) + D_1)}{RTT_1 - RTT_2}$$

Since $\lambda(t) = \alpha t + c$, we obtain after rearranging:

$$\mu = \frac{\alpha}{2} \frac{((t_1 - t_2) - (RTT_1 - RTT_2))((t_1 + t_2) - (RTT_1 + RTT_2) + \frac{2c}{\alpha})}{t_1 - t_2} \tag{6.6}$$

Equation (6.6) shows that the bottleneck capacity μ can be estimated from any two samples of $RTT(t)$.

We now consider the case with multiple sources in a FIFO network. Suppose two sources A and B increase their rates λ_a and λ_b at a same rate α and at time $t = 0$ they have different initial values c_a and c_b, ie. $\lambda_a(t) = \alpha t + c_a$ and $\lambda_b(t) = \alpha t + c_b$. Similarly, we have $\dot{q}(t) = \lambda_a(t - D_1) + \lambda_b(t - D_1) - \mu$. We then obtain:

$$\mu = \alpha \frac{((t_1 - t_2) - (RTT_1 - RTT_2))((t_1 + t_2) - (RTT_1 + RTT_2) + \frac{(c_a + c_b)}{\alpha})}{t_1 - t_2} \tag{6.7}$$

Note that each individual source does not normally have the knowledge as to how many sources are sharing the bottleneck. If each source estimates its share of μ as if it is the only user, ie. based on (6), we get:

$$\mu_a = \frac{\alpha}{2} \frac{((t_1 - t_2) - (RTT_1 - RTT_2))((t_1 + t_2) - (RTT_1 + RTT_2) + \frac{2c_a}{\alpha})}{t_1 - t_2} \tag{6.8}$$

$$\mu_b = \frac{\alpha}{2} \frac{((t_1 - t_2) - (RTT_1 - RTT_2))((t_1 + t_2) - (RTT_1 + RTT_2) + \frac{2c_b}{\alpha})}{t_1 - t_2} \tag{6.9}$$

From (6.7), (6.8) and (6.9), we have

$$\mu_a + \mu_b = \mu \tag{6.10}$$

Equation (6.10) shows that the sum of the μ's estimated by each individual source based on local information is equal to the μ estimated based on global information. It is an important feature that any estimation algorithms to be used in a FIFO network must have. In a FIFO network, global information on the flows sharing one bottleneck is not available to the sources. Each source has to estimate based on local information and feedback. Equation (6.10) assures that, when there are more than one sources sharing a bottleneck, each one can use (6.6) to determine its share of the μ.

While (6.10) guarantees that the individual estimation based on (6.6) can lead to an optimal overall traffic demand, the sharing of the bottleneck resources between different sources are usually not equal. In fact, if $c_a \neq c_b$, we get $\mu_a \neq \mu_b$. The fact that the sharing of μ depends on the initial values c_a and c_b implies the need for coordination between the sources at traffic adjustment in order to achieve fairness.

In order to eliminate the effects of the initial value c, we re-write (6.5) as

$$\sqrt{RTT(t)-D_1-D_2} = \frac{\sqrt{q(t-RTT(t)+D_1)}}{\sqrt{\mu}} \tag{6.11}$$

Suppose that RTT_1 and RTT_2 are two samples of $RTT(t)$, similarly we get:

$$\sqrt{RTT_1-D_1-D_2} - \sqrt{RTT_2-D_1-D_2} = \frac{\sqrt{q(t_1-RTT_1+D_1)} - \sqrt{q(t_2-RTT_2+D_1)}}{\sqrt{\mu}} \tag{6.12}$$

Since $q(t) = \frac{\alpha}{2}(t - D_1 - \frac{\mu-c}{\alpha})^2$, we have:

$$\mu = \frac{\alpha}{2}\left(\frac{(t_1-t_2) - (RTT_1-RTT_2)}{(RTT_1-D_1-D_2)^{1/2}-(RTT_2-D_1-D_2)^{1/2}}\right)^2 \tag{6.13}$$

Note that (6.13) is independent of the initial value c. We can similarly obtain the equations for the case of two sources A and B sharing one bottleneck.

$$\mu = \alpha\left(\frac{(t_1-t_2) - (RTT_1-RTT_2)}{(RTT_1-D_1-D_2)^{1/2}-(RTT_2-D_1-D_2)^{1/2}}\right)^2 \tag{6.14}$$

$$\mu_a = \frac{\alpha}{2}\left(\frac{(t_1-t_2) - (RTT_1-RTT_2)}{(RTT_1-D_1-D_2)^{1/2}-(RTT_2-D_1-D_2)^{1/2}}\right)^2 \tag{6.15}$$

$$\mu_b = \frac{\alpha}{2}\left(\frac{(t_1-t_2) - (RTT_1-RTT_2)}{(RTT_1-D_1-D_2)^{1/2}-(RTT_2-D_1-D_2)^{1/2}}\right)^2 \tag{6.16}$$

From (6.14), (6.15) and (6.16), we get:

$$\mu_a + \mu_b = \mu \tag{6.17}$$

$$\mu_a = \mu_b = \frac{1}{2}\mu \tag{6.18}$$

Equation (6.17) and (6.18) show that the two sources can determine their fair share of the bottleneck capacity based on (6.13) independent of the initial values c_a and c_b as long as they are increasing their traffic rates simultaneously. Nevertheless, (6.13) requires the knowledge of round trip propagation delay D_1+D_2, which is often difficult to obtain.

In general, fairness is difficult to achieve in a FIFO network without any global coordination or network control. Recently, several fair-queueing disciplines have been proposed [Deme89, Zhan89, Rama87] in which packets from different flows are served in a virtual round-robin fashion. In a fair-queueing network, the fair sharing of the network resources is enforced by the switches therefore each source can view its connection over a logical virtual path. As the number of flows sharing the bottleneck changes, each source still need to estimate its current allocation of resources.

6.4.2. Simulation Results

We now verify the analytical results with simulation and discuss the practical issues in applying the estimation techniques into congestion control schemes.

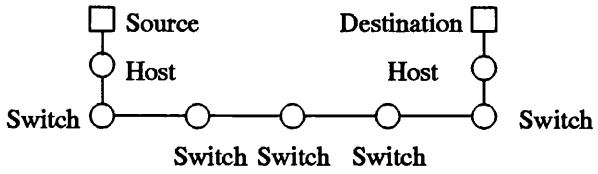


Figure 6.3: Simulation Configuration

The simulation configuration is shown in Figure 6.3. Links between a host and a switch have a capacity of 10 mbps with 1 ms propagation delay. Links between the switches have a capacity of 300 kbps with 50 ms propagation delay. During each estimation session, λ is first set to zero and then increases 5 kbps every 50 ms, ie. α = 100 kbps and c = 0. All packets are of the same size 512 Bytes. The traffic source has a Possion distribution with average traffic rate μ. The first switch is obviously the bottleneck and it has a capacity of 300 kbps.

In the first simulation experiment, we verify the two basic estimation equations (6.6) and (6.13). Each source maintains a list RTT_LIST which records the current 6 consecutive RTT values and their corresponding time when they are calculated, When an acknowledgement is received, the RTT_LIST is updated and the first and the last items in the RTT_LIST list are used to calculate μ based on (6.6) and (6.13).

In (6.13), we need to obtain the round trip propagation delay D_1+D_2. Since the RTT is approximately equal to the round trip propagation delay under light traffic load, we treat the current minimum of the RTTs as the round trip propagation delay. The accuracy of this approximation is discussed later. Each source has a variable RTT_MIN which records the current minimum of RTTs. When a new RTT is calculated and is smaller than RTT_MIN, RTT_MIN is updated. Since λ is increased from zero, there is a period when the traffic load is very light before the queue at the bottleneck starts to build up. The RTTs obtained during this period are very close the round trip propagation delay.

Figure 6.4 shows the evolution of the estimated μ based on (6.6) and (6.13) when one source increases its λ linearly. In Figure 6.4(a), the estimated μ increases linearly under light traffic load and levels off around $t = 3.7$ s. It then remains around the bottleneck capacity 300 kbps. In contrast to Figure 6.4(a), Figure 6.4(b) shows that the estimated μ decreases sharply when the traffic load increases (the figures are too large to be shown when $t < 3.5$ s). However, the estimated μ eventually converges to the bottleneck capacity 300 kbps around $t = 3.7$ s. Figure 6.4 confirms our analytical results that fluid model approximation and estimation equations (6.6) and (6.13) are accurate when the traffic load is high.

The time $t = 3.7$ s represents the turning point when the packets start to build up at the bottleneck. This can also be verified in Figure 6.4(c) which records the RTT value at the time when μ is estimated. After $t = 3.7$ s, RTTs start to increase rapidly as the queueing delay increases. In theory, the estimation of μ needs only two RTT samples after $t > 3.7$ s. However, the accuracy of the estimation may be improved if a number of estimation can be carried out.

Since the estimation is only valid when the packets start to build up at the bottleneck, we need some mechanisms to detect the turning point when estimation can start. In the second simulation experiment, we use a simple mechanism for detecting the turning point. When a RTT is calculated, it is compared with RTT_MIN. If the difference exceeds 25 ms (equivalent to the queueing delay of two 512 bytes packets over a 300 kbps link), the estimation of μ is repeated five times and the average is considered as the estimated bottleneck capacity. The simulation is

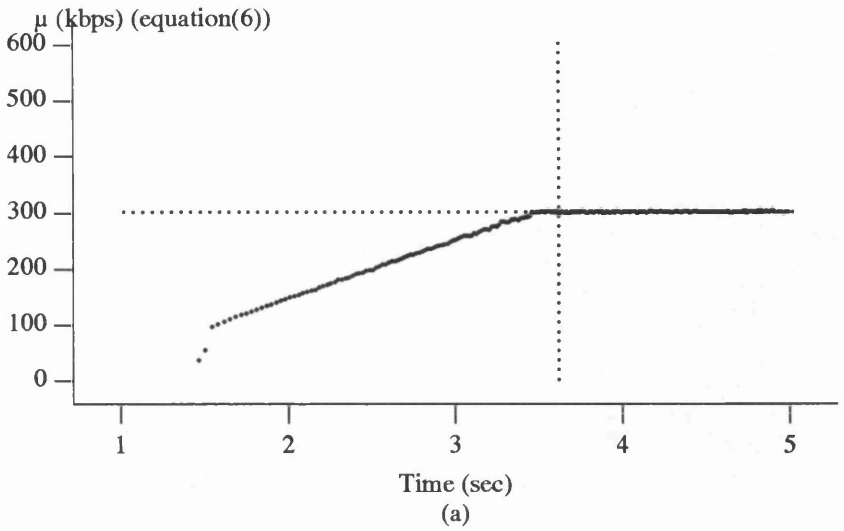
also repeated 15 times on each of the 10 settings with the bottleneck capacity varying from 100 kbps to 1000 kbps. The results (see Table 6.1) show that with the simple mechanism to detect the turning point, the estimated μ is a very good approximation of the actual bottleneck capacity.

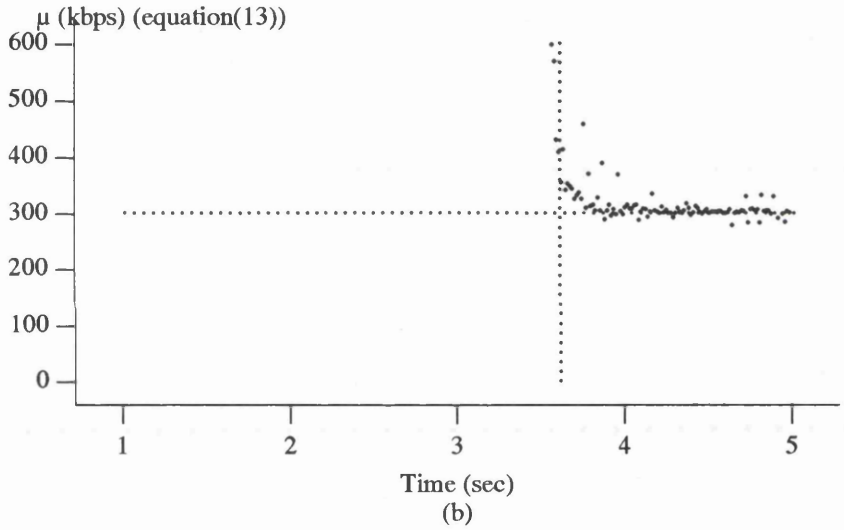
Bottleneck Capacity (kbps)	Estimated μ (equation(6)) (kbps)	Estimated μ (equation(13)) (kbps)
100	97.7	96.8
200	198.2	197.3
300	298.5	297.2
400	399.1	398.7
500	500.2	499.8
600	602.7	601.4
700	704.3	701.1
800	803.5	802.3
900	904.5	903.3
1000	1005.4	1004.8

Table 6.1: Average Results of Estimated μ

6.5. Summary

The mechanisms for information feedback in congestion control are investigated. Two different types of approaches are identified. Two new binary feedback schemes and one quantitative feedback scheme are proposed. The simulation shows that the analytical result based on the fluid model is accurate.





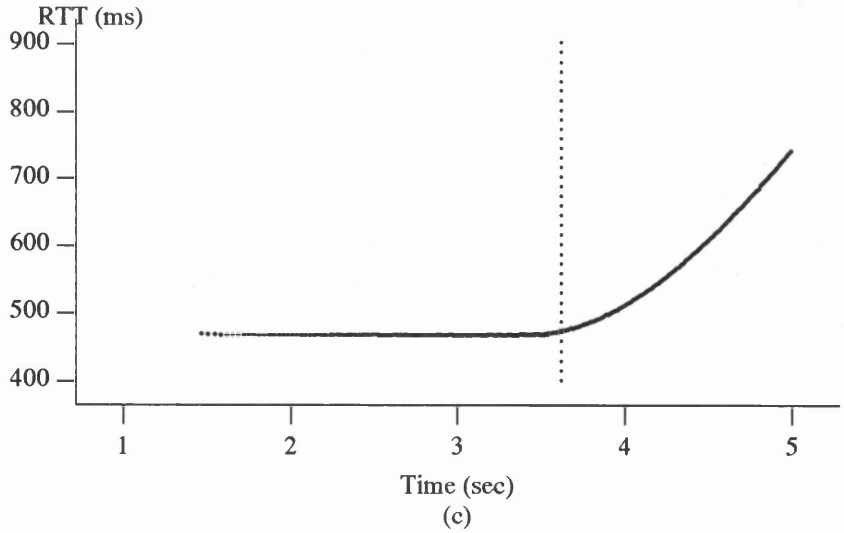


Figure 6.4: Evolution of Estimation μ

CHAPTER SEVEN

END-SYSTEM TRAFFIC ADJUSTMENT

This chapter presents a new traffic adjustment algorithm called *synchronized traffic adjustment (STA)* and three end-system congestion control schemes based on STA and the information feedback schemes described in the previous chapter. Simulation results show that the new congestion control algorithms can speed up the convergence and minimize the oscillation of traffic.

7.1. Introduction

Most of the proposed end-system congestion control schemes are based on the additive increase and multiplicative decrease (AIMD) algorithm for traffic adjustment, although the information feedback mechanisms used are different. Due to the recursive and continuous nature of the adjustment, the AIMD algorithm often requires many iterations to reach equilibrium so convergence is usually slow. Long propagation delay in the feedback can also cause clear oscillating behaviors [Zhan89, Bolo90]. Note that there is at least one round trip time latency in the information feedback. Suppose that at time t source A is increasing or decreasing its demand. The information as to whether the demand of source A at time t is optimal or not will not be available to source A until time $t+RTT$. During this period, source A will continue to increase or decrease its demand. Bolot and Shankar [Bolo90] have shown that the magnitude of such oscillation is proportional to D^2, where D is the round trip propagation delay.

In this chapter we present a new traffic adjustment scheme called *synchronized traffic adjustment (STA)* and three congestion control schemes based on the STA approach and the information feedback schemes described in the previous chapter. The performance of the three congestion control schemes are studied extensively by simulation. The simulator used is a modified version of the MIT Network Simulator [Heyb89, Mart88]. The topology used in the simulation is

shown in Figure 7.1.

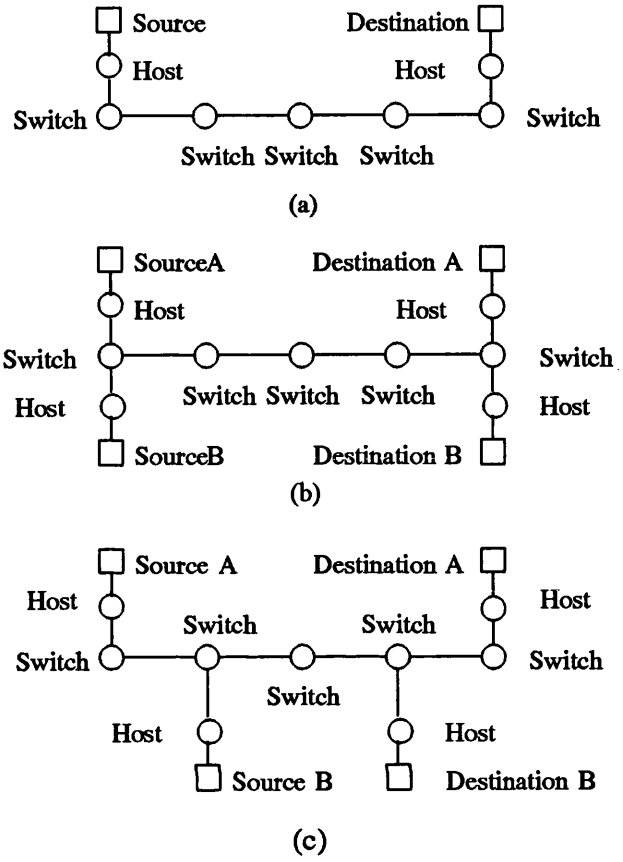


Figure 7.1: Simulation Topology

7.2. Synchronized Traffic Adjustment

STA adopts a very different approach compared to the additive increase and multiplicative decrease scheme. STA attempts to establish the sharing of the resources when there are significant traffic changes, eg. at the beginning and the end of a connection. Once the sharing of the resources has been settled, it will remain unchanged until new changes occur. We feel that it is not possible for end-systems to adapt to dynamic traffic fluctuations because of the delay in information feedback, and temporary traffic fluctuations have to be dealt with by buffering and network queueing algorithms. The fairness of the sharing in STA is enforced by synchronization of traffic adjustment. When a significant traffic change is detected, a signal is fed back to all the users sharing the bottleneck resources. All users start to search for their new operating points by increasing the traffic from a pre-specified level, until another signal is received.

There are three types of signals: *decrease*, *increase* and *stay*. A decrease signal indicates that a significant traffic increase has been detected and all users should reduce their traffic rates. An increase signal indicates that a significant traffic decrease has been detected and all users can increase their traffic rates. A stay signal indicates that the current traffic level is optimal and therefore invites users to stay with the current traffic rates.

As an example, we describe the basic operations involved when a user joins in and terminates.

- When a user starts a connection, it enters the initial phase. The user increases its traffic rate from a basic unit to the maximum allowed by the destination. This increase is to force the current users to start an adjustment session. If the bottleneck resources can not accommodate the traffic increase, a decrease signal is then sent to all users sharing the bottleneck resources and all users enter the decrease phase.
- In the decrease phase, all users start to decrease their traffic rates. The actual decrease is done by resetting the traffic rates to a pre-agreed level and then to increase simultaneously. Since the new user often has less share of the resources when the decrease session starts, the reset is the very mechanism to ensure that the current users do not have any advantages over the new user and all current users start the search from the same starting point. The reset also serves as a slow start to allow congestion to be eased. When a stay signal is received, all users enter the stay phase and wait for new signals.
- When an increase signal is received, all users enter the increase phase and start to increase their traffic rates, absorbing the released resource. Since all current users have an approximately equal share of the resources, a reset is not necessary and all users can increase their rates from the current levels. When a stay signal is received, all users enter the stay phase.

7.3. Rate-Based Traffic Adjustment

In this section we present a rate-based congestion control scheme based on STA and equation (6.13) derived in chapter six. Our intention is to demonstrate the basic mechanisms used in the STA algorithm and to compare it with the AIMD algorithm.

We assume that the sources generate traffic at the specified transmission rates. To simplify the model, we ignore the error recovery procedures and focus on the rate adjustment aspects. In other words, lost packets are not retransmitted. When a new connection starts, it increases from λ_0 at the rate α until it receives a decrease signal. It then resets the rate to λ_0 and starts to increase at

the rate α. During the increase, the source estimates its share of the bottleneck resources. When it receives a stay signal, the source sets the transmission rate to the he current estimate of the bottleneck rate. When an increase signal is received, the source starts to increase its λ from the current value and sets the transmission rate to the current estimate when a stay signal is received.

The signals are carried in two signal bits in the packet header. The queue length Q at switches are monitored and compared with two thresholds Q_{lower} and Q_{upper}. When Q is below Q_{lower}, the two signal bits are set to 01 (increase signal). When Q is above Q_{upper}, the two signal bits are set to 10 (decrease signal). Otherwise the two signal bits are set to 11 (stay signal). When the destination receives a packet, it copies the two signal bits into the acknowledgement packet.

The topology used in the simulation is shown in Figure 7.1(b). Links between a host and a switch have a capacity of 10 mbps with 1 ms propagation delay. Links between the switches have a capacity of 1 mbps with 50 ms propagation delay. All packets are of the same size of 512 bytes. The queue length thresholds for signaling Q_{upper} and Q_{lower} are 2 packets and 10 packets respectively. The interval for rate adjustment is 500 ms, which is chosen to be slightly larger than the round trip propagation delay so that the effect of the current adjustment can be checked in the next round traffic adjustment. When traffic rates are being increased, the rate α is 100 kbps/s. The increase actually occurs once in each adjustment interval when the queue length is checked, ie. a 50 kbps increase for every 500 ms.

Figure 7.2 shows the result of the traffic adjustment when a new user joins in the existing user at time t=0 and leaves at time t=24. When both users receives the decrease signal, their transmission rates are reset to 50 kbps and then increases 50 kbps every 500 ms. Around t=6, both users receive the stay signal and set their rates to the current estimates 470 kbps and 480 kbps respectively. Around t=25, one user starts to increase as a result of the increase signal until a new stay signal is received.

To compare the performance, we also simulate the AIMD algorithm with the same settings. We replace the STA algorithm in the sender with the additive increase and multiplicative decrease algorithm. The sender increases its traffic rate by 50 kbps when it receives an increase signal and decreases by 1/8 of current value otherwise. This is equivalent to setting the congestion bit when the queue length is over Q_{lower} = 2 [Rama88]. The interval for traffic adjustment is also 500 ms. The result (Figure 7.3) shows that the convergence of the AIMD algorithm is much slower as compared with that of the STA algorithm. While the two users in Figure 7.2 take about 6 seconds to settle the sharing of the resources, the two users in Figure 7.3 take more than 18

seconds to reach such a point. With the STA algorithm, the rate α can be further increased to speed up the adjustment. With AIMD, a large α can lead to a significant increase in the magnitude of the oscillation. The behaviors of the two algorithms when one user leaves are similar.

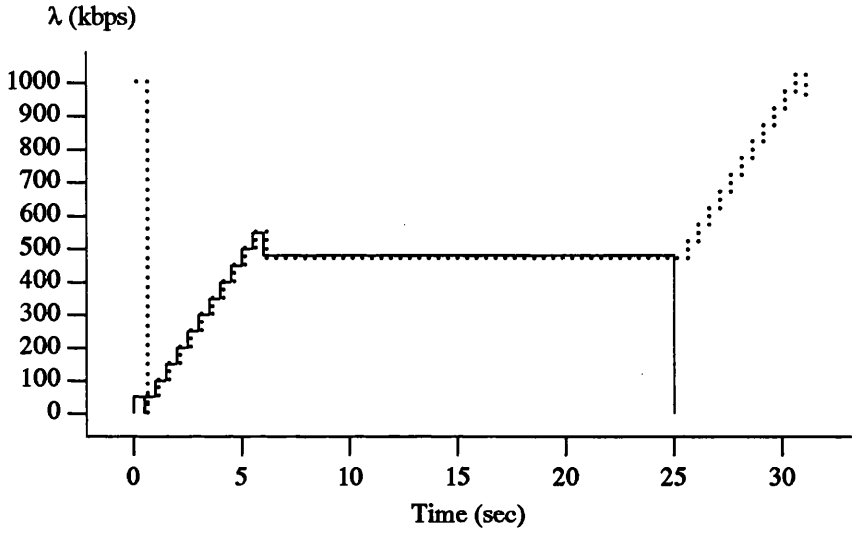


Figure 7.2: Rate Adjustment With the STA scheme

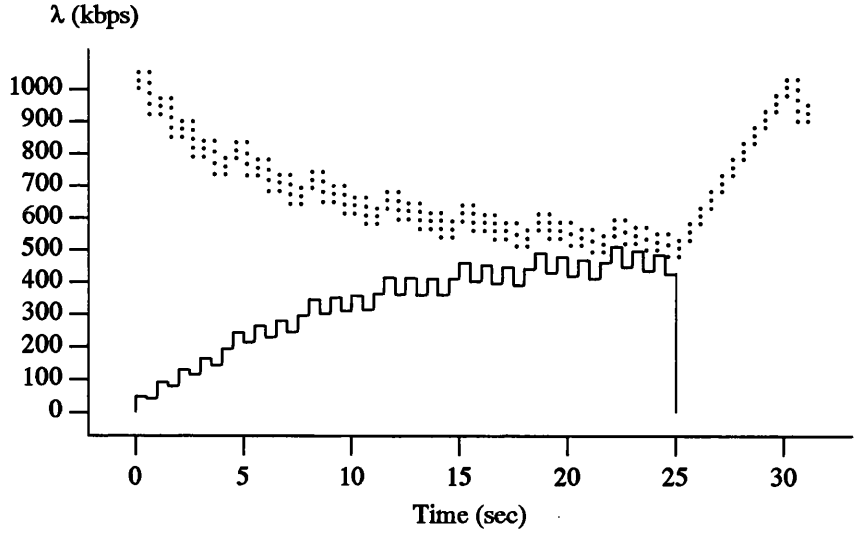


Figure 7.3: Rate Adjustment With the AIMD scheme

7.4. Window-Based Traffic Adjustment

In this section we present two window congestion control schemes based on the STA algorithm and the binary feedback schemes discussed in the previous chapter. The two schemes are implemented into the TCP module of the MIT simulator and tested in various settings. Our aims are to eliminate the traffic oscillation and packet losses problems associated with the BSD 4.3 tahoe TCP congestion control mechanism.

7.4.1. Tri-S

Slow Start and Search (Tri-S) is a congestion control scheme based on the STA traffic adjustment algorithm and the throughput gradient feedback scheme. It uses the timeout as the decrease signal and derives the stay and increase signals from the normalized throughput gradient (NTG).

The operation of the Tri-S scheme can be described in the three phases of the STA algorithm:

- When a user starts a connection, it is in the initial phase. The window size is set to one. When an acknowledgement is received, it increases the window size by one, until the maximum window size allowed by the destination is reached. In the initial mode, the window opens exponentially until it reaches the maximum window size or detects a timeout.
- When a timeout is detected, it enters into the decrease phase. The window size is reset to one. When an acknowledgement is received, the *NTG* is checked. If the *NTG* is larger than a threshold NTG_d, the window size is increased by one. Otherwise it enters the increase phase. In the decrease phase, the window also opens exponentially but switches to linear increase once the *NTG* becomes smaller than NTG_d.
- In the increase phase, the window size is increased by 1/(window size) for each acknowledgement received. When the accumulated increase is larger than one, the *NTG* is checked. If the *NTG* is smaller than a threshold NTG_i, the window size is decreased by one. In the increase phase, the window opens linearly until the *NTG* reaches the threshold NTG_i.

We now discuss the computation of *NTG* in the decrease and increase phases. Note that the *NTG* can be expressed as:

$$NTG(W_n) = \frac{TG(W_n)}{TG(W_1)}$$

$$TG(W_n) = \frac{T(W_n)-T(W_{n-1})}{W_n-W_{n-1}}$$

where W_n and W_{n-1} represent the two sequential window sizes and $T(W_n)$ is the throughput at window size of W_n.

The average throughput during the round trip time of the *nth* packet can be expressed by: $T(W_n) = \frac{W_n}{D_n}$, where W_n is the number of bytes outstanding in the network (including the packet being transmitted) at the time of transmission of the *nth* packet and D_n is the round trip delay measured when the acknowledgement of the *nth* packet is received. The window size is adjusted each time by one so we have

$$W_n - W_{n-1} = W_1 - W_0 \quad \text{and} \quad NTG(W_n) = \frac{T(W_n) - T(W_{n-1})}{T(W_1)}$$

Let W_{now} and D_{now} represent the current window size and the current round trip delay respectively, measured at the time when an acknowledgement is received, and window size and delay have been updated. If all users have the same packet size, W_n can be derived directly from W_{now}. The relationship between W_n and W_{now} in the increase mode is illustrated as (W_{now} , W_n) in Figure 7.4. The number of packets outstanding in the network changes only when the window size is increased, ie. when an acknowledgement is received. When the acknowledgement of the *nth* packet is received, there are n packets outstanding in the network, the window size increases from n to n+1 and the (2n)*th* and (2n+1)*th* packet are sent out. So we have

$$W_n = \frac{W_{now}}{2} \quad \text{and} \quad T(W_n) = \frac{W_{now}}{2D_{now}} \quad (\textit{if } n \textit{ is an odd number})$$

$$W_n = \frac{W_{now} - 1}{2} \quad \text{and} \quad T(W_n) = \frac{W_{now} - 1}{2D_{now}} \quad (\textit{if } n \textit{ is an even number})$$

In the increase mode, the window opens linearly and the *NTG* is evaluated only when the window size is increased by one. We get

$$W_n = W_{now} - 1 \quad \text{and} \quad T(W_n) = \frac{W_{now} - 1}{D_{now}}$$

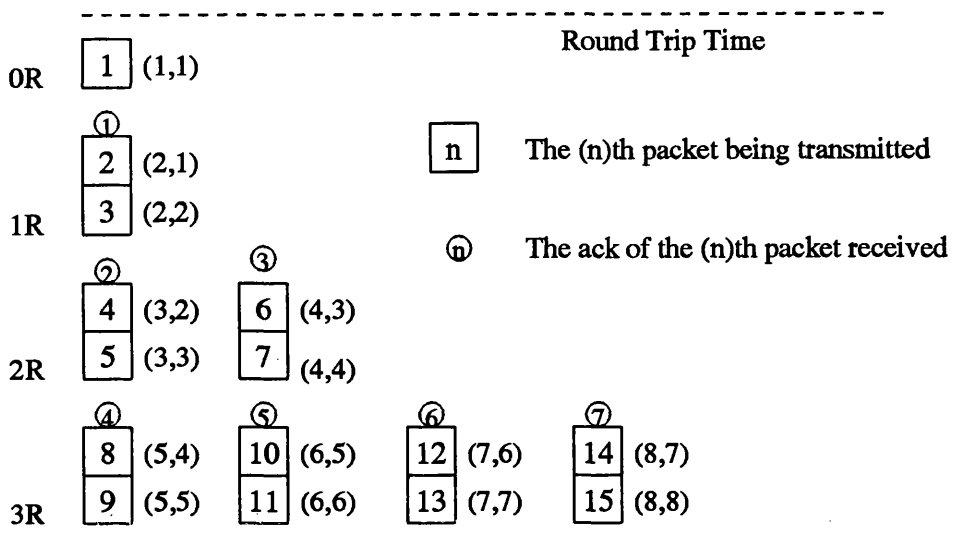


Figure 7.4: The Relationship Between the (n)th Packet and Its Acknowledgement

The sharing of the bottleneck resources settled by the Tri-S scheme depends on the timing of the starting point of the decrease phase. To ensure fairness implies the need for synchronization at the beginning of the decrease phase. In the Tri-S scheme, the decrease phase is triggered by the packet losses and timeouts. It has been observed that the losses of packets by the overflow of the bottleneck buffer and the retransmissions of individual flows are highly synchronized [Zhan89].

We now present the simulation results. In our simulation experiments, there are five carefully selected simple scenarios:

- One user using one path,
- Two users sharing one path,
- Two users sharing one path partially,
- One user joining in a steady flow,
- One user leaving a steady flow.

The transit behavior is important here in order to understand and visualize the operation of the scheme. In our experiments, we plot the data of a particular run and also present the average results of 20 independent runs.

Figure 7.1(abc) shows the basic topologies used in the simulation. All packets are of the same size 512 bytes. Links between a host and a switch have a capacity of 1000 kbps with zero

propagation delay. Links between the switches have a capacity of 500 kbps with 50 ms propagation delay. Each switch has a buffer of 30 packets. The maximum window size allowed by the receiver is 85 packets. The thresholds NTG_i and NTG_d are 0.5. We assume that the data transmission lines are error-free. For each scenarios, both the Tri-S scheme and the Jacobson's scheme are tested. The average results of 20 independent runs are shown in Table 7.1 (the first 10 seconds of data is excluded). For each scenario, samples of the results in the first 40 seconds of one particular run are also plotted (see Figure 7.5 - Figure 7.14). In the first two scenarios, the window size is plotted instead of the throughput as they are equivalent and the window size can show more clearly the operation of the scheme. The throughput is measured every time when an acknowledgement is received. The queue length refers to the outgoing queue of the first switch.

		Tri-S		Slow-Start	
		average	variance	average	variance
Scenario One	Throughput (kbytes/s)	57.90	1.12	52.43	2.31
	Queue Length (pks)	2.33	0.23	12.12	4.12
Scenario Two	Throughput (User A) (kbytes/s)	29.11	1.12	31.43	3.41
	Throughput (User B) (kbytes/s)	29.43	1.38	26.81	4.31
	Queue Length (pks)	2.11	0.33	10.59	3.72
Scenario Three	Throughput (User A) (kbytes/s)	23.71	5.38	13.71	4.42
	Throughput (User B) (kbytes/s)	34.11	7.19	36.61	3.66
	Queue Length (pks)	5.12	3.01	13.81	5.08
Scenario Four	Throughput (User A) (kbytes/s)	24.50	5.30	18.62	3.46
	Throughput (User B) (kbytes/s)	36.13	6.20	31.96	3.77
	Queue Length (pks)	8.32	3.14	12.73	3.76
Scenario Five	Throughput (User A) (kbytes/s)	59.01	5.10	35.55	4.83
	Queue Length (pks)	1.81	0.44	3.17	3.54

Table 7.1: Average Results

A. Scenario One

In the simplest configuration, ie. a link with one flow (see Figure 7.1(a)), the Tri-S scheme demonstrates some of the most desired features of traffic control (see Figure 7.5). After initial negotiation period, the window size stablizes. It can be seen from the queue length graph that the

bottleneck operates at its full capacity while the queuing delay remains very low. A closer look at both the window size graph and the queue length graph reveals some interesting details: from 10 seconds onwards, each time when the queue length drops from two to one, the window size increases one to bring the queue length back to two. Figure 7.6 shows that apart from the slow-start period the Jacobson's scheme oscillates the window size between the maximum and half the maximum.

B. Scenario Two

In this scenario, two users share the same link (see Figure 7.3(b)). This scenario is to test the behavior of the schemes in the presence of competition for resources. Although the average results of the two schemes are relatively close, the samples of data plotted in Figure 7.7 and Figure 7.8 show significant difference between two schemes. Many of the problems with Jacobson's scheme discussed previously are illustrated here clearly. The Tri-S scheme shows stable and optimal behaviors. The window sizes of the two users are extremely close and the variations are small.

C. Scenario Three

When two users only partially share one path (see Figure 7.3(c)), the Tri-S scheme can no longer achieve exact equal sharing of the resource between the two users. The reasons are discussed later in the section. Nevertheless, the queue length is still stable and short and the sharing of resources is relatively fair as compared with the Jacobson's scheme. The oscillating behavior of the Jacobson's scheme is aggravated in this scenario as the user with shorter delay changes the window size more rapidly. There are some black areas in the throughput graphs instead of curves. This is caused by the high density of the data.

D. Scenario Four

In the previous scenarios, the two users start their connections at the same time. In this scenario, one of the two users that share one link partially (see Figure 7.3(c)) starts its connection first and the other user joins in 9 seconds later. It is shown that the final results are similar to those in the scenario three.

E. Scenario Five

In the final scenario, one of the two users that share one link partially terminates its connection. The Tri-S scheme detects the change in traffic quickly and moves to another optimal operating point. Jacobson's scheme can also absorb the released resources but the window size will be

increased until a timeout and start the oscillation again. This can be been seen in Figure 7.14 as it will take long time for the queue to build up.

The Tri-S scheme presented here is still very primitive and needs to be further improved. We now discuss some important issues and areas for future research.

The thresholds NTG_i and NTG_d are important parameters. NTG_d determines the operating point at which the flows will settle during the demand adjustment session. Experiments show that when NTG_d is smaller than 0.3 or larger than 0.8, the average queue length at the bottleneck is often above 20 or below 0.5. The value we use in the simulation (0.5) maintains the queue length between 2 and 10. NTG_i affects the sensitivity of detecting the released resources. It should be large enough to allow some fluctuation of traffic flows. The precise relationship between the thresholds and operating point has to be further studied.

The round trip delay of the first packet in the slow-start (D_1) plays an important role in operating-point search. It eliminates the effects of different round trip delays and therefore normalizes the NTG. It is, in fact, approximately equal to the propagation delay as it is the round trip delay with the smallest window size (1). It is highly desirable that the first packet of a TCP connection (the segment with the SYN bit set) has a higher priority, so that the effects of queuing delay can be minimized and propagation delay can be measured accurately. In our simulation, however, the effects of queuing delay are surprisingly small. It is due to the fact that D_1 is the round trip delay of the first packet measured after a timeout when the path is nearly empty. Nevertheless, it is still desirable to measure the propagation delay with a higher priority packet. In a complex traffic conditions, it may not be possible to ensure that there is no traffic after the timeout period. Propagation delay is an important parameter that can be of great use in routing and traffic control yet it is not available in most current transport protocols.

The synchronization between users at the starting point of the decrease phase affects the fairness of the sharing. The synchronization of packet losses and timeouts used in the Tri-S scheme weakens when the users have different round trip delays. The unfairness when two users share partially is caused by the fact that two users have different timeout periods. The user that has the shorter delay tends to retransmit first and therefore has a larger share. The packet clustering phenomenon in window control systems also has a negative effect on the synchronization. It has been shown that some network algorithms such as Fair Queueing can improve the mixture of packets from different flows [Shen90].

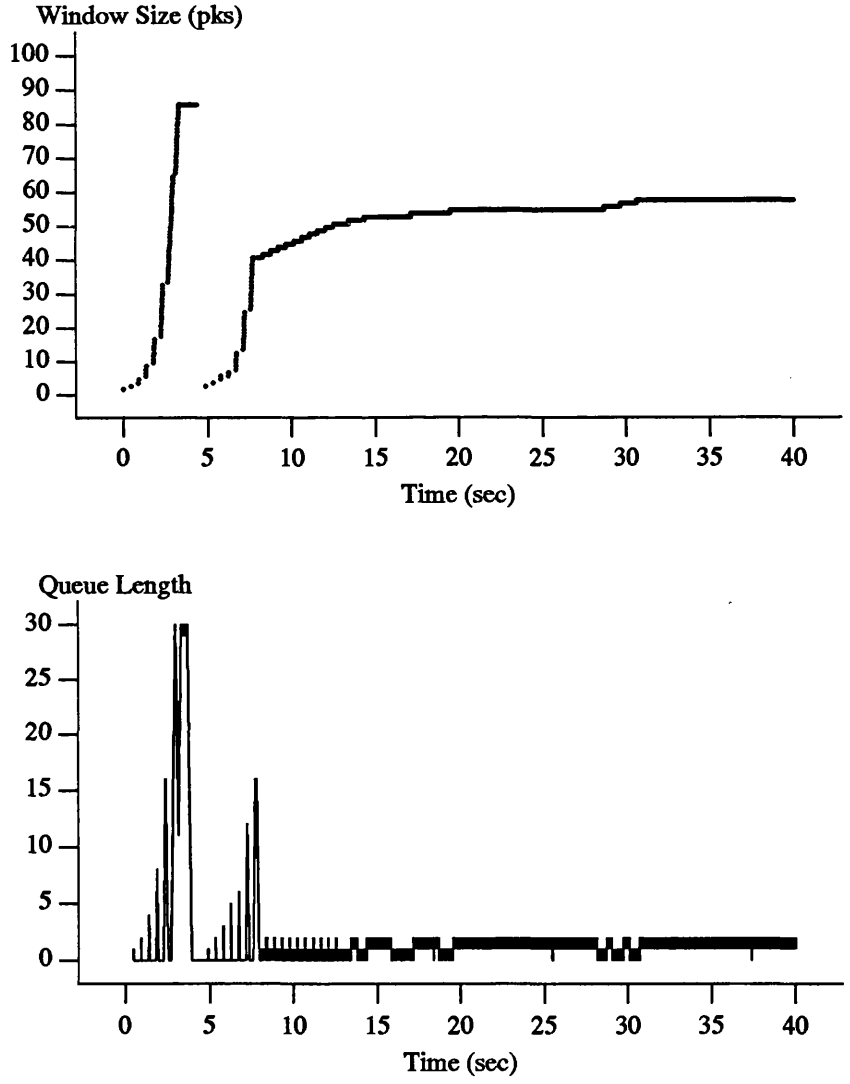


Figure 7.5: Scenario One (Tri-S)

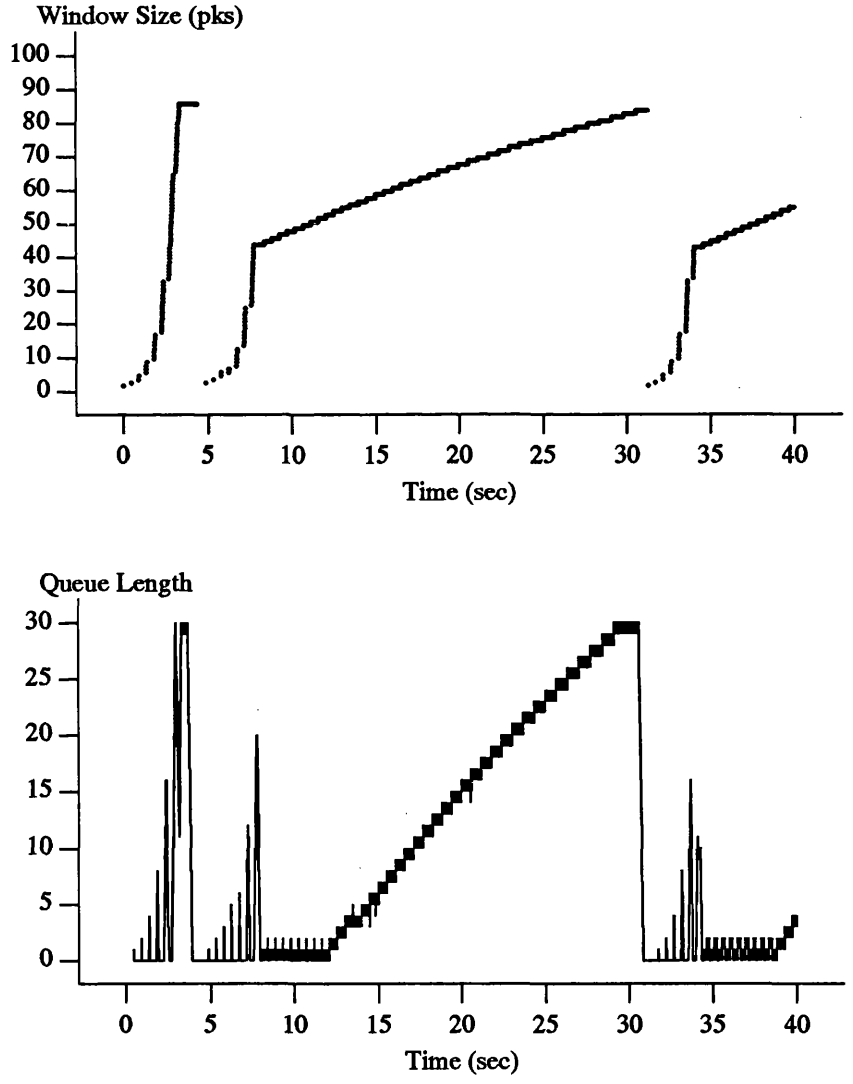
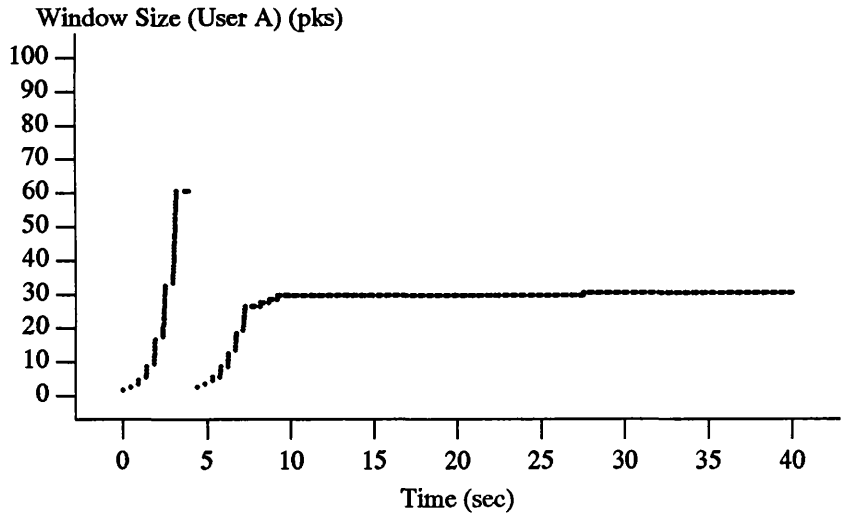
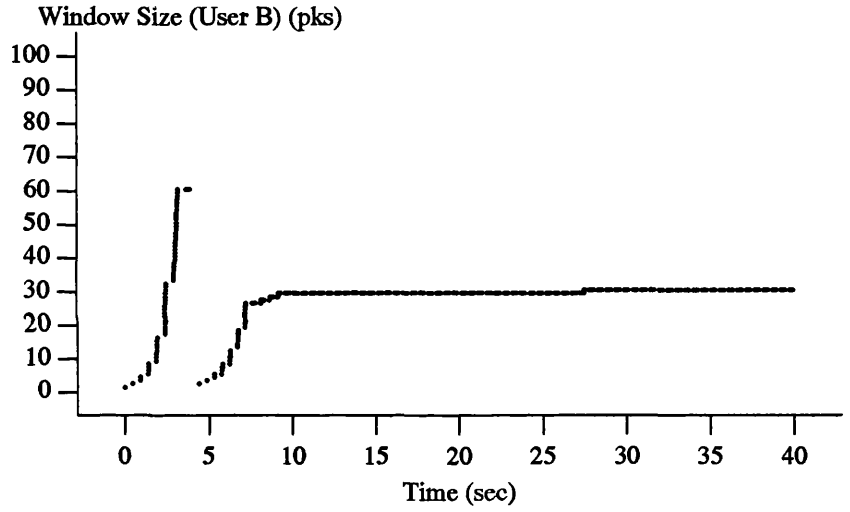


Figure 7.6: Scenario One (TCP tahoe)





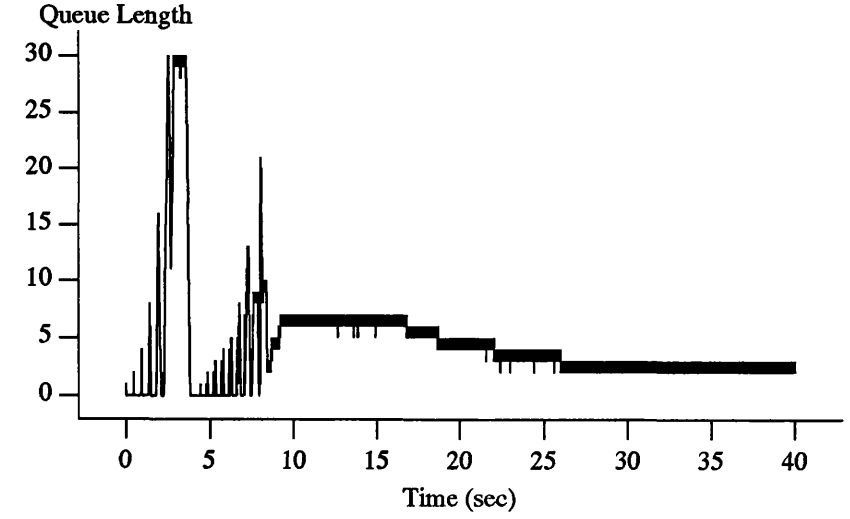


Figure 7.7: Scenario Two (Tri-S)

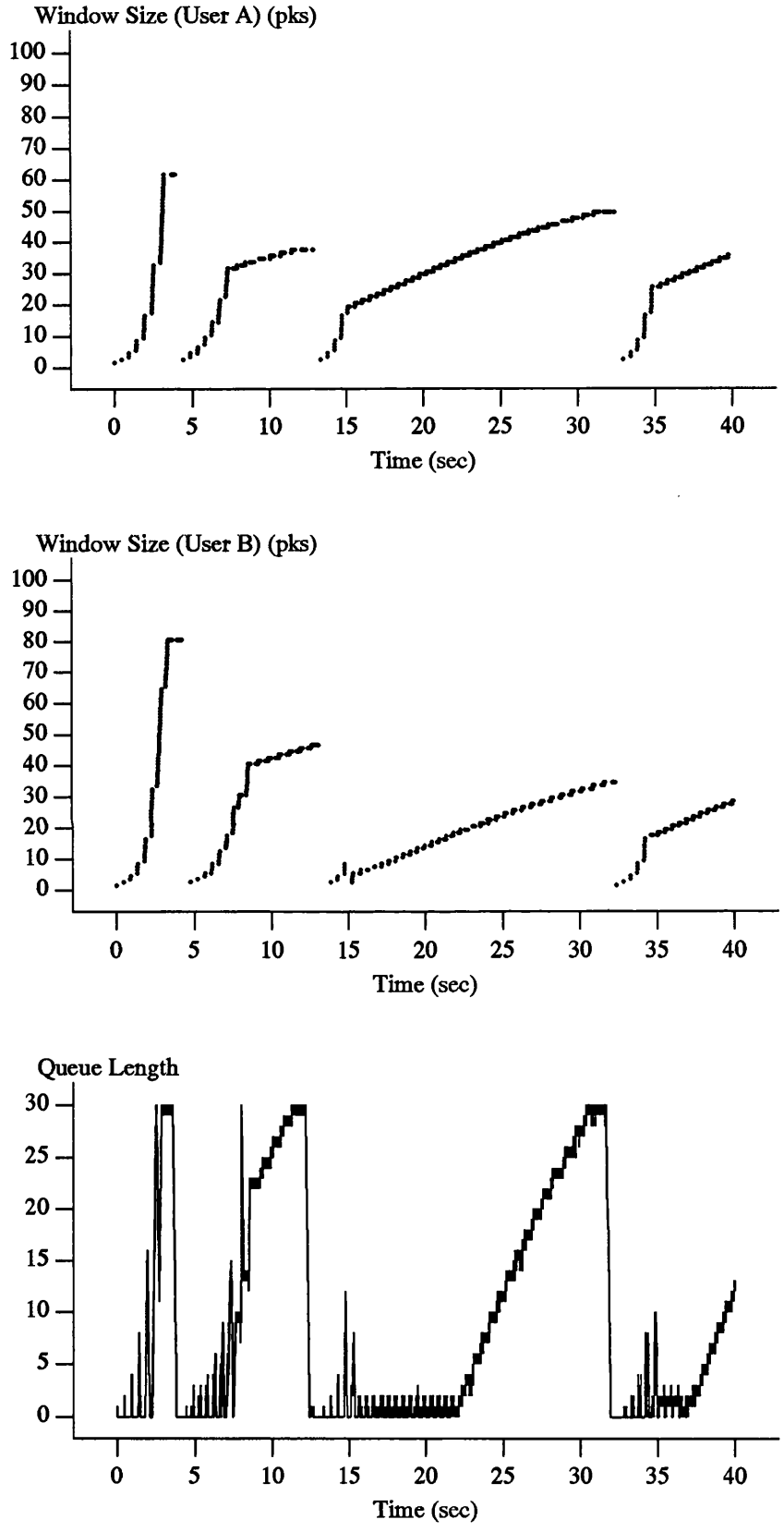
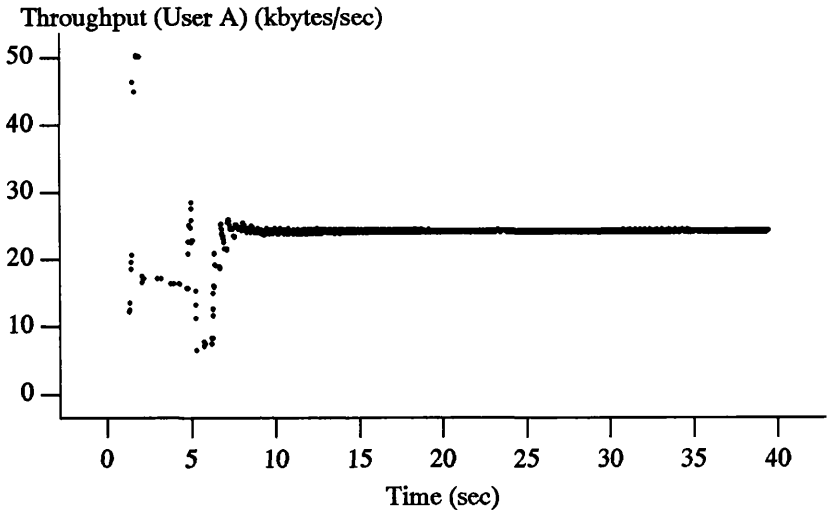
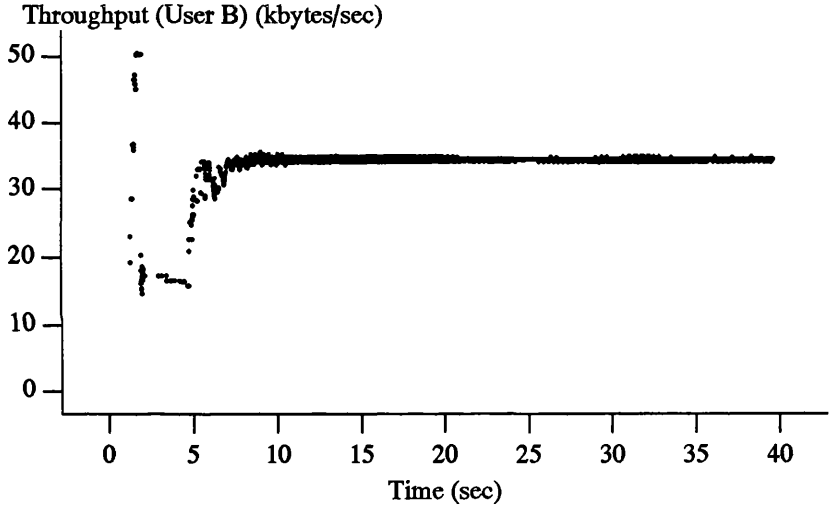


Figure 7.8: Scenario Two (TCP tahoe)





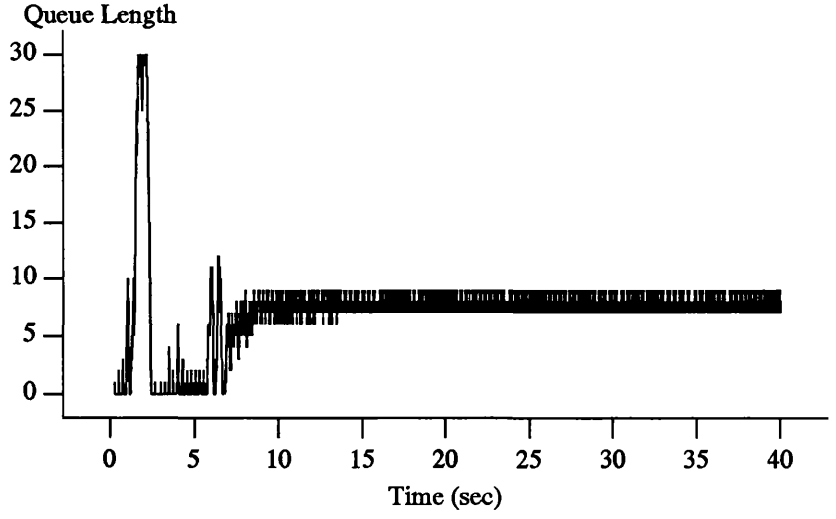
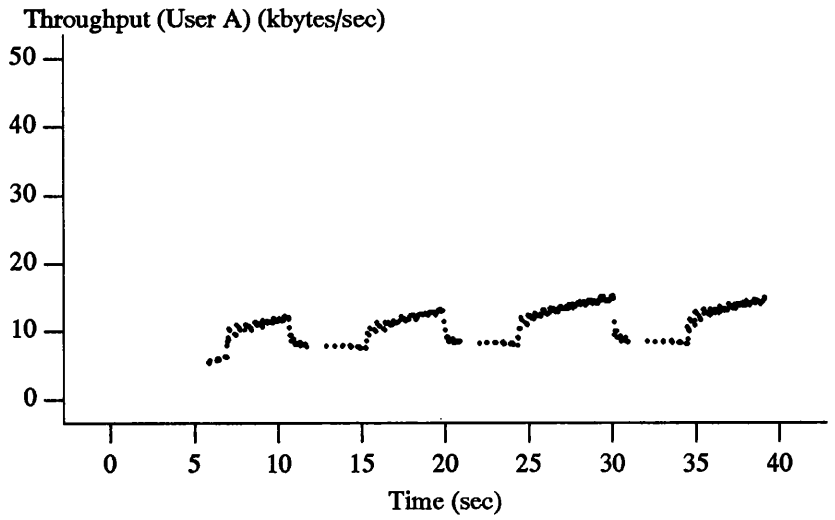
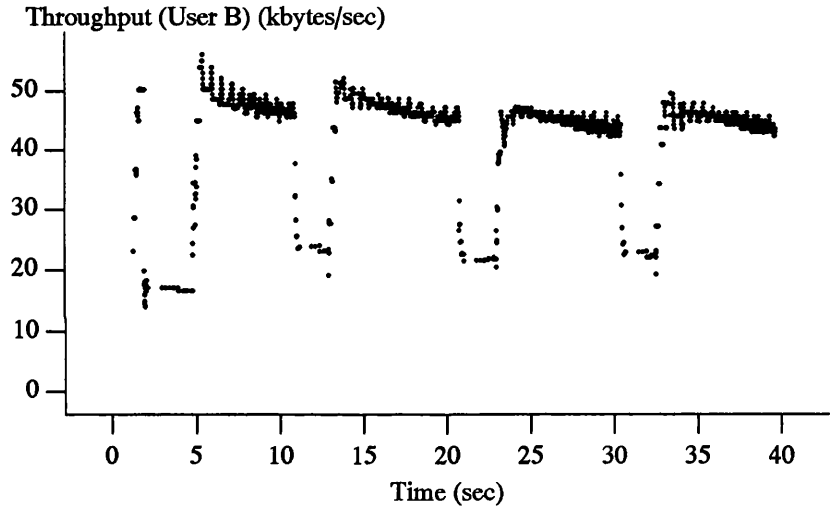


Figure 7.9: Scenario Three (Tri-S)





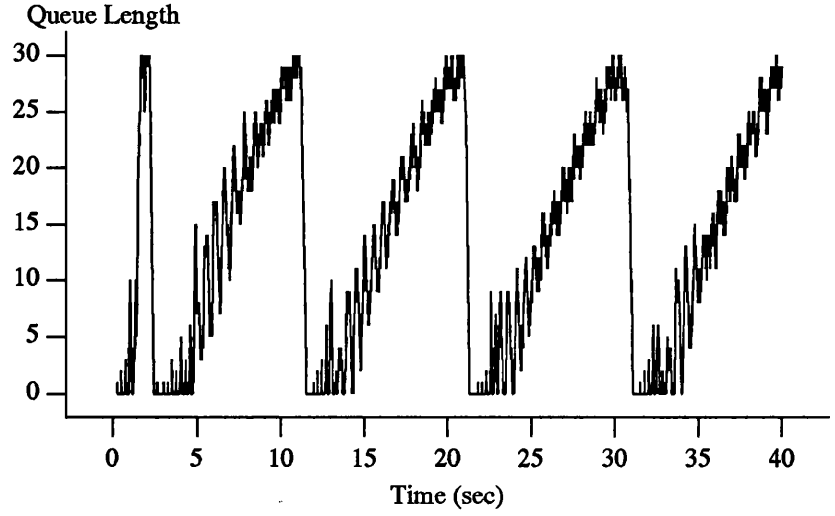


Figure 7.10: Scenario Three (TCP tahoe)

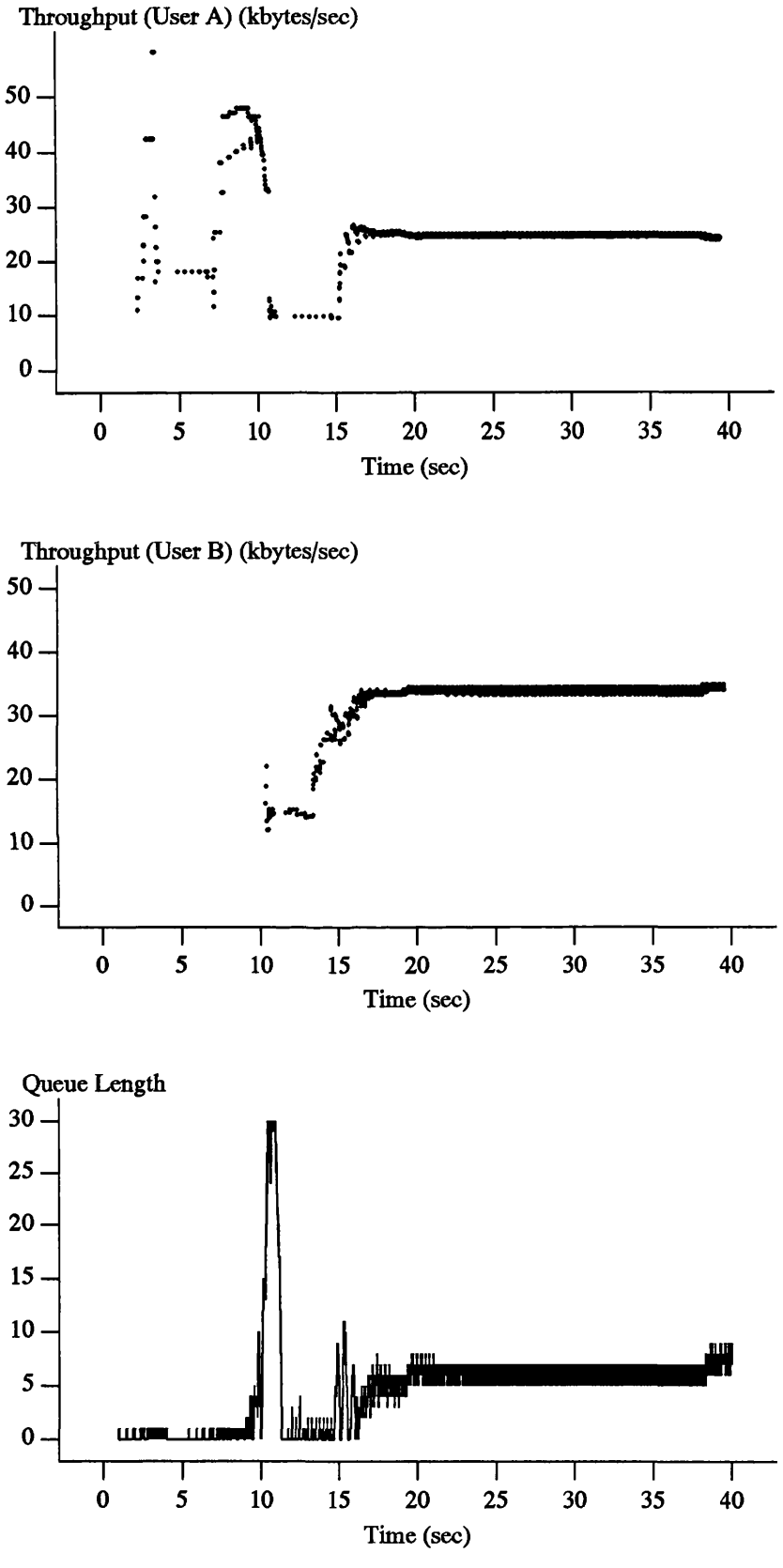


Figure 7.11: Scenario Four (Tri-S)

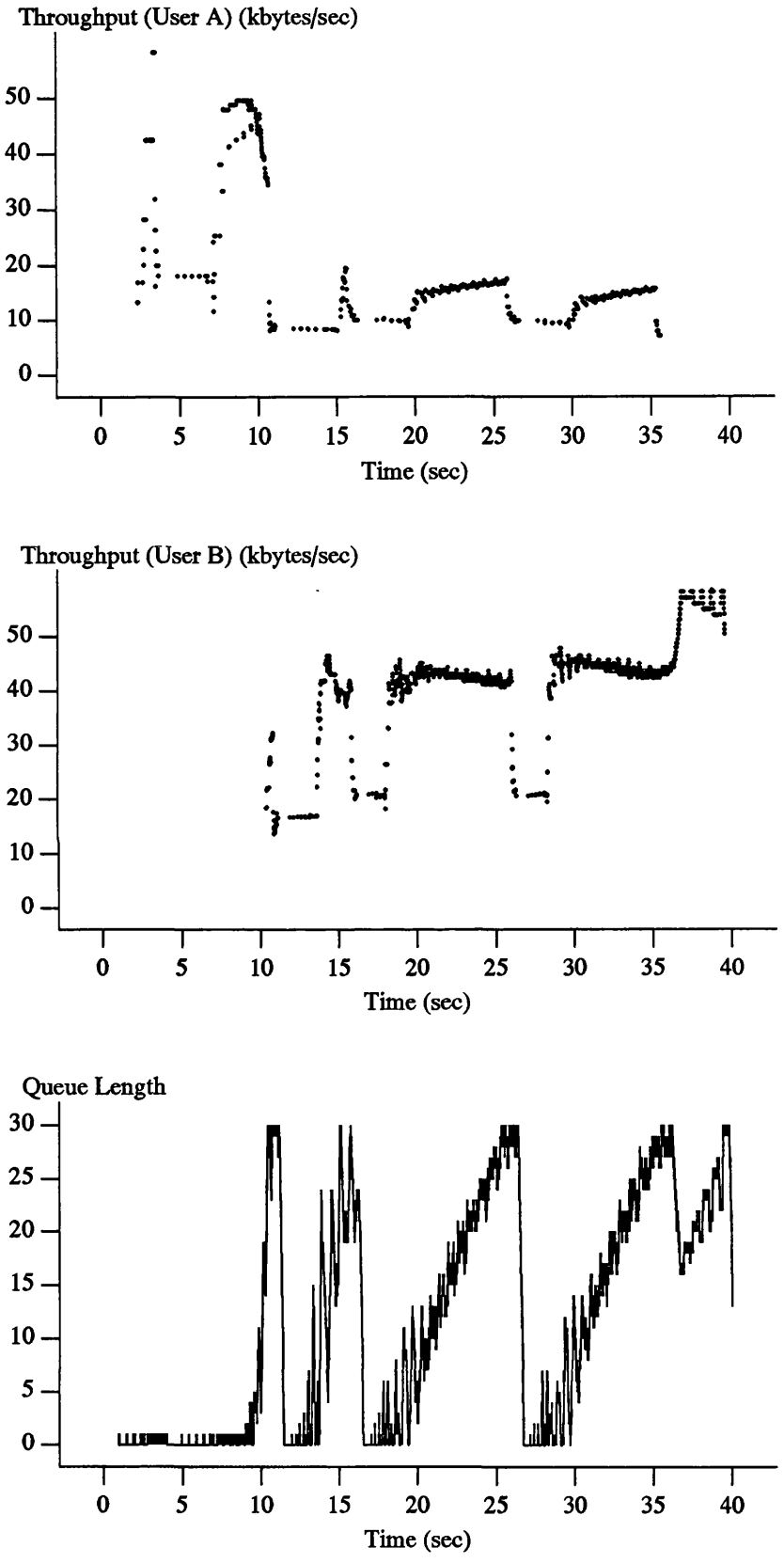
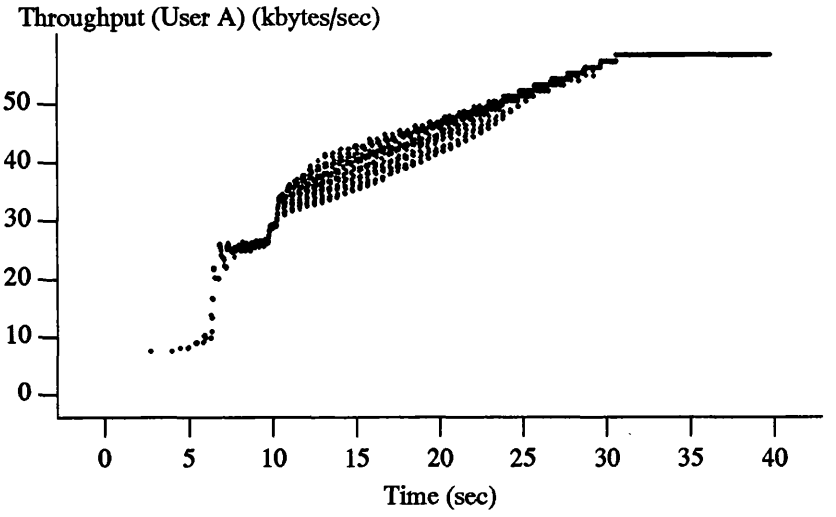
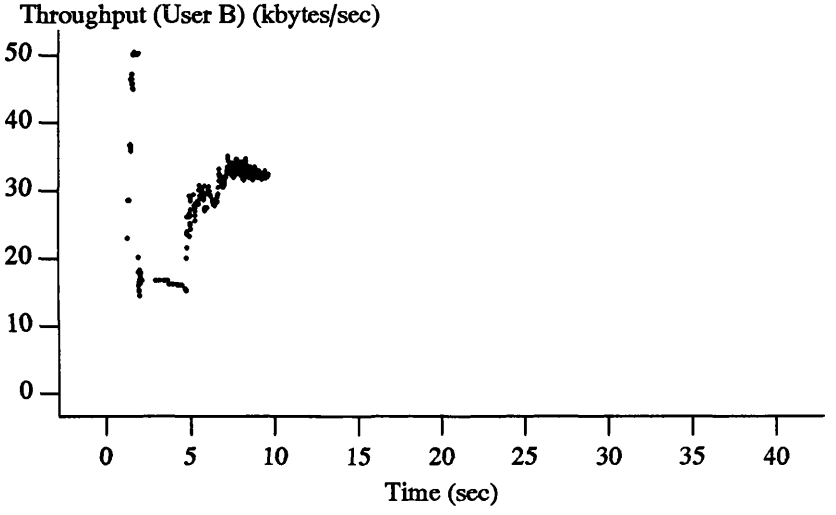


Figure 7.12: Scenario Four (TCP tahoe)





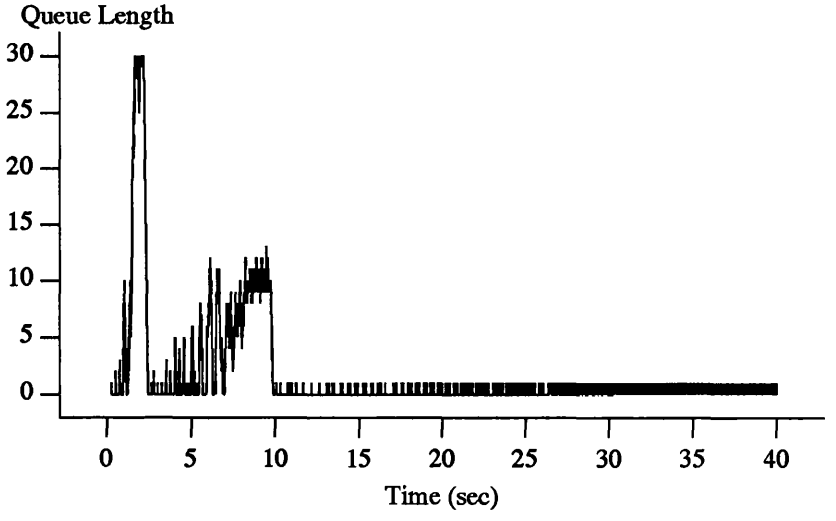


Figure 7.13: Scenario Five (Tri-S)

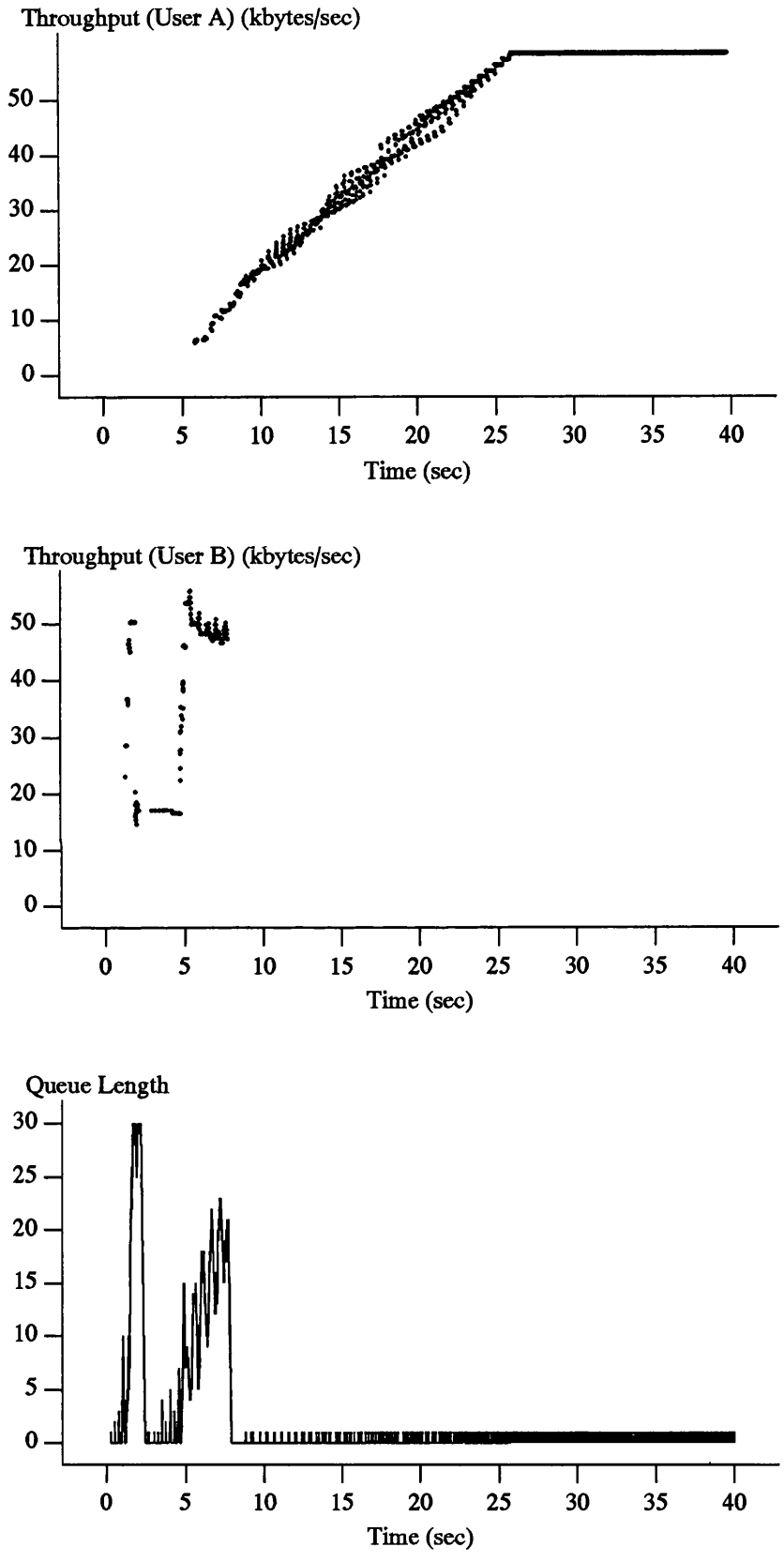


Figure 7.14: Scenario Five (TCP tahoe)

7.4.2. DUAL

The DUAL congestion control scheme is a combination of the timeout and delay threshold feedback schemes and the STA and AIMD traffic adjustment algorithms.

In the lifetime of a connection, an end system has to deal with both major traffic changes such as the beginning and ending of a connection, and minor traffic changes such as resource probing at the source or natural fluctuation of the traffic source. Most of the congestion control schemes do not distinguish the two different changes and treat them the same. The result is either slow convergence or wild oscillation. Take the AIMD algorithm for an example, to speed up the convergence will lead to increase in the oscillation. The STA algorithm attempts to detect and adjust to major traffic changes and leave minor fluctuations to the switches. The optimal solution is to differentiate the two traffic changes and deal with them separately.

In the DUAL scheme, we use both timeout and delay threshold as congestion signals and STA and AIMD for traffic adjustment to achieve two levels of adjustment: *quick-tuning* and *fine-tuning*.

We now describe the implementation of the DUAL scheme in the BSD 4.3 tahoe TCP congestion control mechanism.

The end system detects the two different traffic changes using two different feedback schemes. When a timeout is detected, the end system assumes that substantial traffic increase and severe congestion have occurred. It then uses quick-tuning to reduce the window size. The slow-start threshold is set to half of the current window and congestion window is set to one. The quick-tuning adjustment is similar to the one used in TCP tahoe [Jaco88]. During normal resource probing phase, ie. window size increases by 1 for each round trip delay, the end system uses delay threshold and fine-tuning to adjust its window size. For every two round trip times, the round trip delay is checked. If $D > D_i$, the congestion window is set to 7/8 of current size. The fine-tuning adjustment is very similar to the DECBIT scheme [Rama88].

When a new connection is set up, the sender does

```
cwnd = 1;
ssthresh = maxwnd;
rtt_min = INF;
rtt_max = 0;
```

When a new RTT measurement is obtained, the sender does

```
rtt_min = min(rtt, rtt_min);
rtt_max = max(rtt, rtt_max)
```

When an acknowledgement is received, the sender does

```
if (cwnd < ssthresh)
  cwnd += 1;
else
  cwnd += 1/cwnd;
```

For every two windows' worth of packets transmitted, the sender does

```
if (rtt > (rtt_min + rtt_max)/2
    && cwnd < ssthresh)
  cwnd -= min(cwnd, wnd)/8;
```

When a timeout is detected, the sender does

```
ssthresh = max(min(cwnd, wnd)/2, 2);
cwnd = 1;
rtt_min = INF;
rtt_max = 0;
```

Finally the sender always does

```
wnd = min(cwnd, maxwnd);
```

We now present the simulation results of the DUAL algorithm and compare them with that of Jacobson's algorithm.

The simulation configuration is shown in Figure 7.1(a)(b). Links between a host and a switch have a capacity of 10 mbps with 1 ms propagation delay. Links between the switches have a capacity of 800 kbps with 10 ms propagation delay. All packets are of the same size 512 Bytes and each end of the TCP connection has a maximum buffer size of 50 packets. The bottleneck is obviously the first switch on the left and it has a capacity of 800 kbps.

A. Scenario One

In this scenario, there is only one user over the path (see Figure 7.1(a)). TCP tahoe opens the connection with a slow-start and then enters the oscillation phase (see Figure 7.15). Packet losses occur at the peaks of the window fluctuations. Although the initial adjustment is almost identical, the DUAL algorithm does reduce the oscillation substantially (see Figure 7.16). The fine-tuning adjustment based on the delay threshold prevents the window from reaching the level of overflow.

B. Scenario Two

In this scenario, two users share one path (see Figure 7.1(b)). One user starts its connection earlier and the other joins in 2.4 seconds later. This scenario is to test the ability of the algorithms to adjust traffic after a new user joins in. From the results shown in Figure 7.17 and Figure 7.18, we can see that the behaviors of the two algorithms are similar to those in the Scenario One. A close look at Figure 7.18 reveals some interesting details. Soon after the two users settle to their new window sizes (around t=5.3 seconds), the window sizes of user A and B are around 12 packets and 21 packets respectively. As the time elapses, the two window sizes move closer (around 16 packets and 19 packets when t=14 seconds). This is exactly what is expected: the timeout triggers a drastic adjustment and then the delay threshold makes some fine adjustment.

C. Scenario Three

Two-way traffic has more complex dynamics than one-way traffic [Zhan91]. In this scenario, we examine the effects of two-way traffic on the DUAL algorithm, in particular, the effects of rapid queue fluctuations on the delay threshold based adjustment. In this scenario, users A and B transmit data in the opposite directions and the results are shown in Figure 7.19 and Figure 7.20 (the queue length shown is that of the first switch on the left in Figure 7.1(b)). Although the DUAL algorithm can not eliminate the rapid queue fluctuations in two-way traffic which is caused by the mixture of data and acknowledgement packets [Zhan91], a well-behaved traffic adjustment does improve the performance. Furthermore, the delay threshold based adjustment is not greatly affected by the large magnitude of queue fluctuations, as one might have expected. The reason is that the actual delay curve is much smoother than the queue length curve because acknowledgement packets which cause the queue length fluctuation are of very small size and therefore the actual queueing delay change is smooth. One detail worth mentioning is that after the two users timeout around t=1 second, the increase in window size is very sharp (almost a straight line), ie. there is no sign of slow-start. This is caused by compressed acknowledgements. When the two connections timeout at t=1 second, there are a large number of

acknowledgement packets blocked by the data packets in the switch queue. Once those acknowledgement packets have been released, each of the acknowledgement packets triggers two new packets. Therefore the window increases sharply which leads to another timeout.

We now discuss some practical issues in the implementation of the scheme into TCP.

The accuracy of D_{min} and D_{max} affects the final window sizes and the final queue length when convergence is reached. It is extremely difficult to have accurate measurements of round trip propagation delay unless there is only one flow over the path. When there are more than one flow over the path, the measured D_{min} is often greater than the actual round trip propagation delay and the queue length is likely to be over the desired operating point $Q_{max}/2$ and will increase as the number of flows increase. Different flows may obtain different values of D_{min}, which may lead to unfairness in sharing the bandwidth. In our simulation we found that the current minimum round trip delay is a simple and reasonable approximation of the round trip propagation delay. This is because that the traffic load during the slow start after a timeout is usually very light therefore the queueing delay is fairly low. We also found that the D_{max} obtained from the packets just before buffer overflow is reasonably accurate and the variation of D_{min} is comparatively large.

When a network control algorithm such as fair queueing [Deme89] is implemented, the measurement of D_{min} and D_{max} can be much improved since each flow is logically separated from others. If we send the first two packets with sufficient gap between them, the first packet will then enable a new queue to be set up at each switch and the second packet will get best possible priority at each switch.

The parameter α determines the final queue length when convergence is reached. The buffers in the switches accommodate the fluctuation of the traffic. Ideally the amount of packets stored in the queue should be just enough to fill the gaps in the packet stream and therefore maximum room can be left for the traffic bursts. Nevertheless, this optimal operating point depends on the characteristics of the traffic, which is unpredictable and time-varying. Half of the maximum queue length is a reasonable choice if the buffer is not very large.

The delay threshold based adjustments also need accurate RTT measurement. The current resolution of TCP delay measurement of 500 ms is inadequate and any improvement in the RTT estimation algorithms is of little use. However, the proposed TCP extensions [Jaco90] will increase the resolution to 1-100 ms, which is sufficient for most circumstances.

7.5. Summary

A new traffic adjustment algorithm *synchronized traffic adjustment (STA)* is proposed. STA attempts to establish the sharing of the resources only when there are significant traffic changes and to maintain fairness by synchronized adjustment. Three congestion control schemes, based on the STA approach and the information feedback schemes discussed in the previous chapter, are presented. Extensive simulation shows that the STA algorithm eliminates the oscillation problems in the AIMD algorithm and converges more quickly.

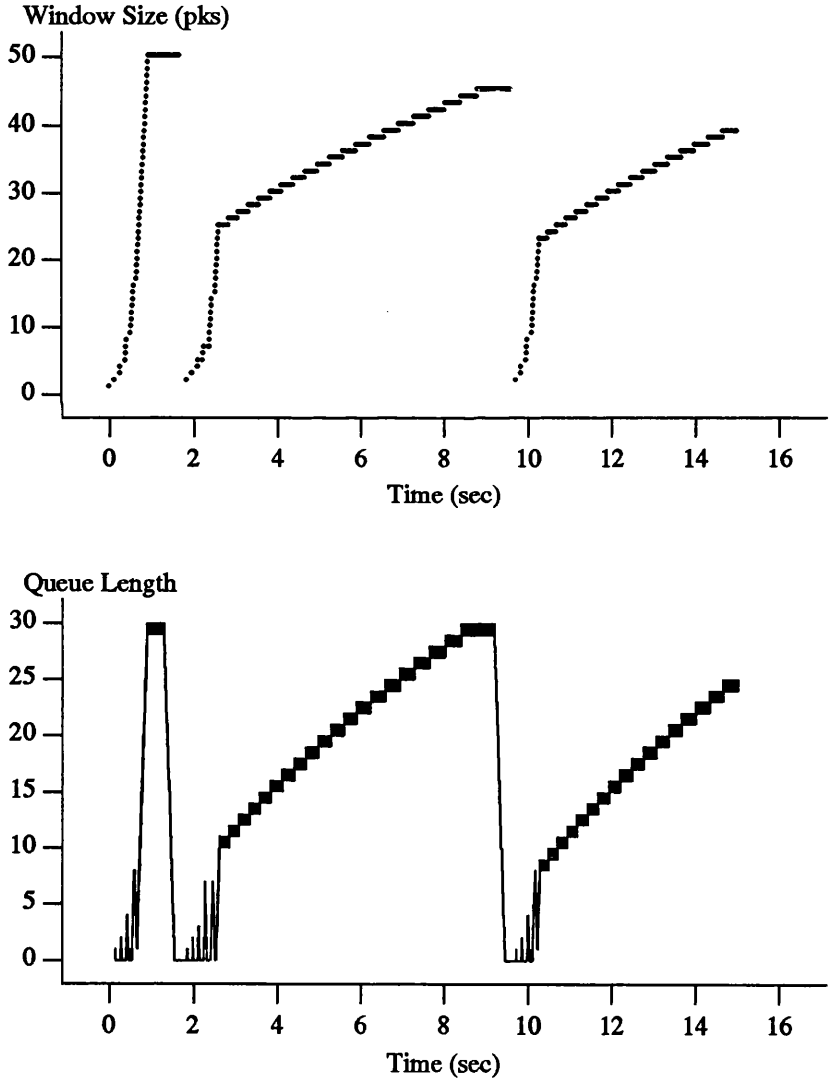


Figure 7.15: Scenario One (TCP tahoe)

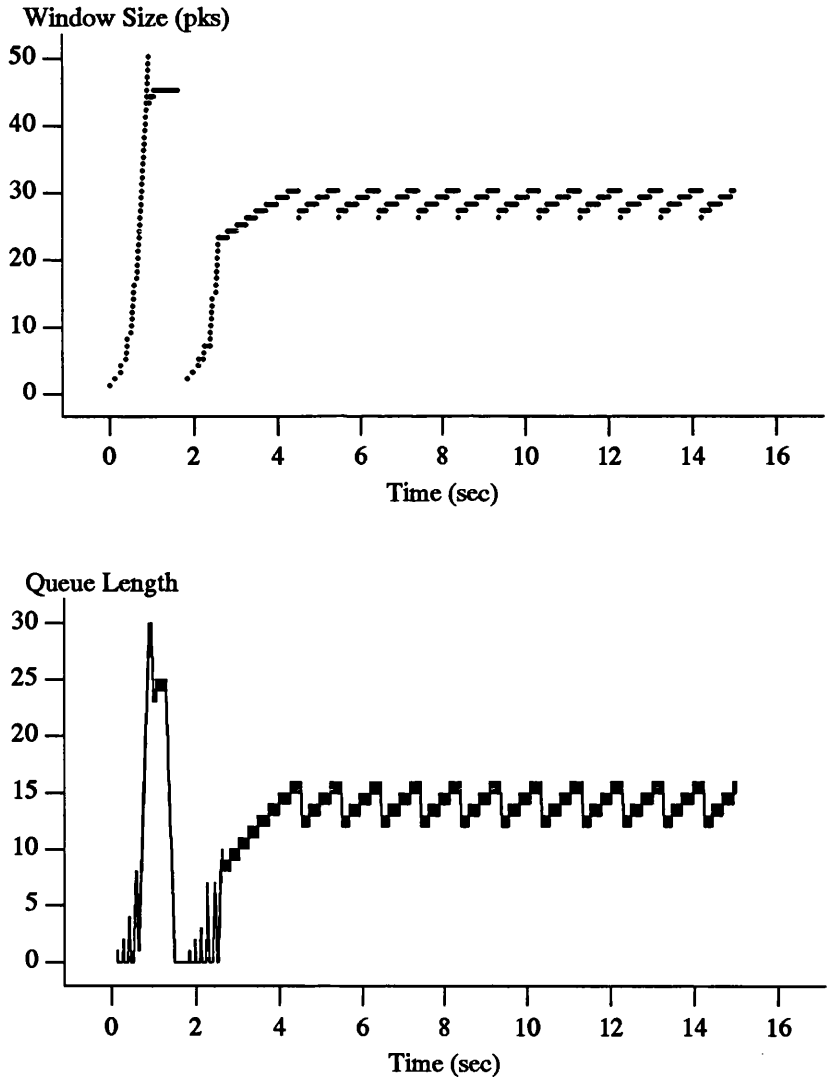
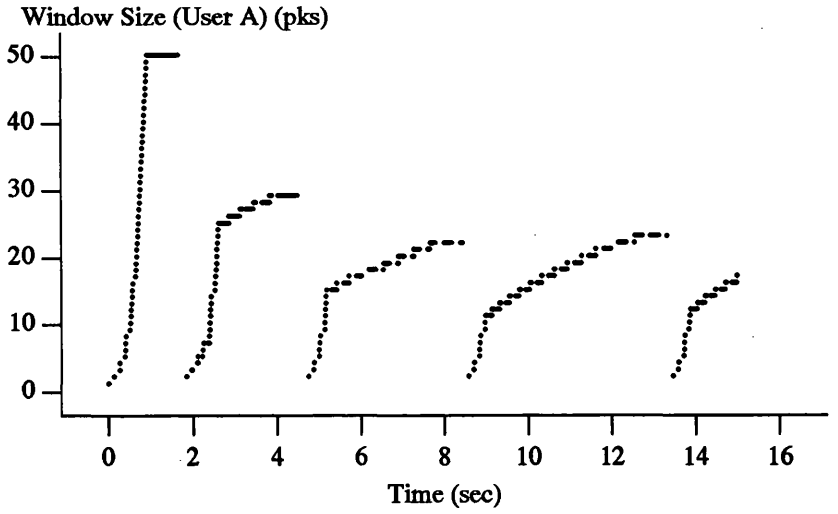
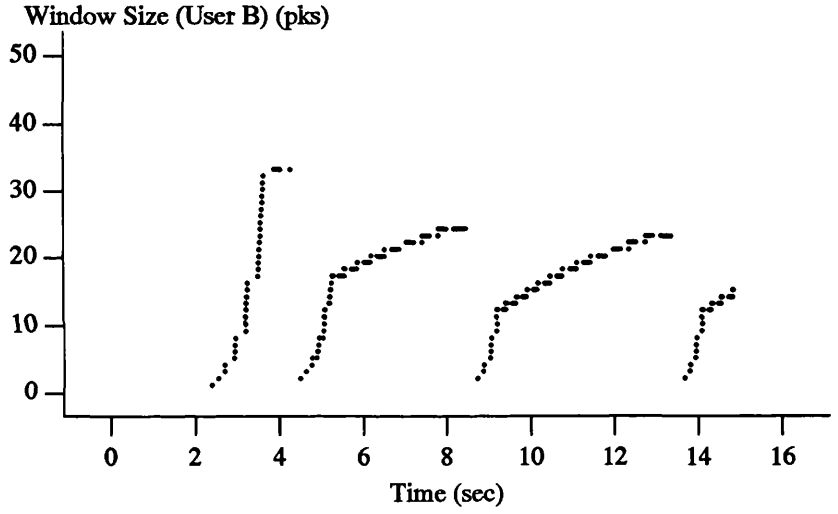


Figure 7.16: Scenario One (DUAL)





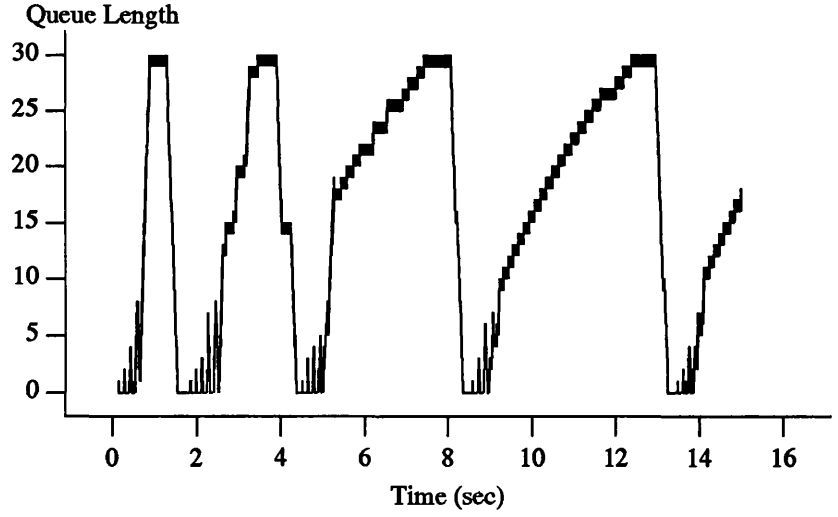


Figure 7.17: Scenario Two (TCP tahoe)

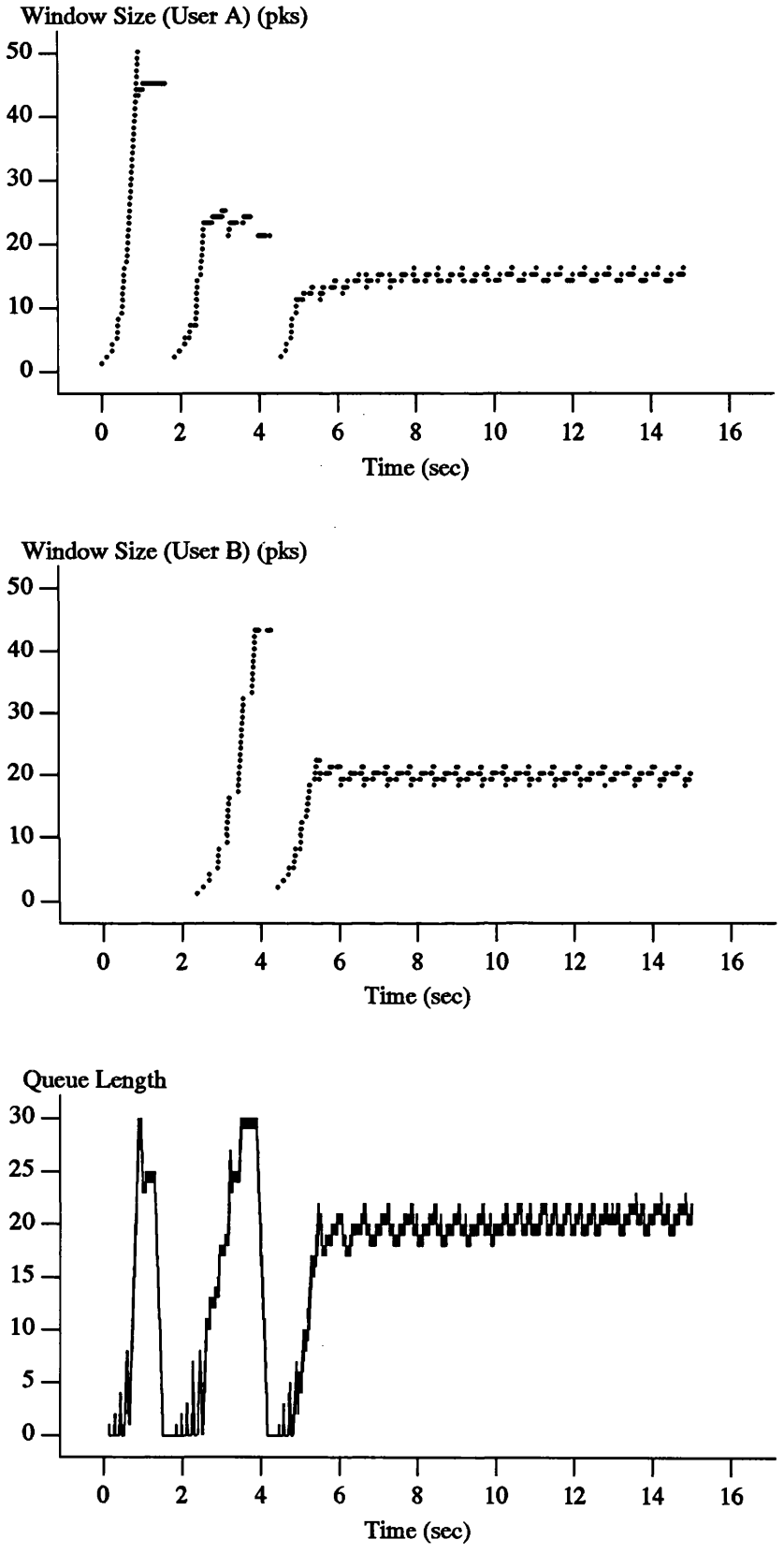
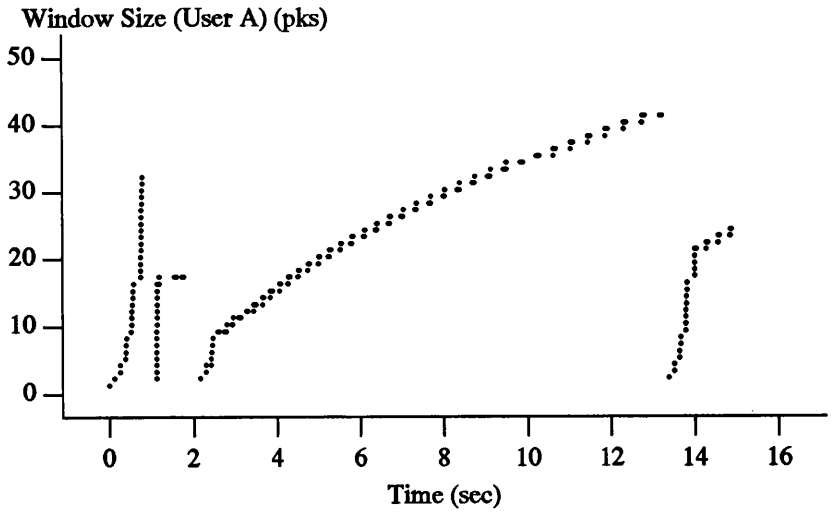
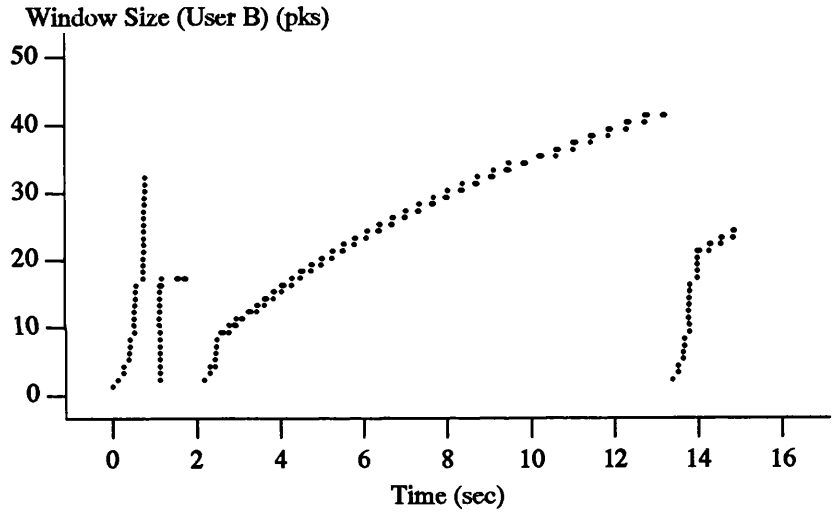


Figure 7.18: Scenario Two (DUAL)





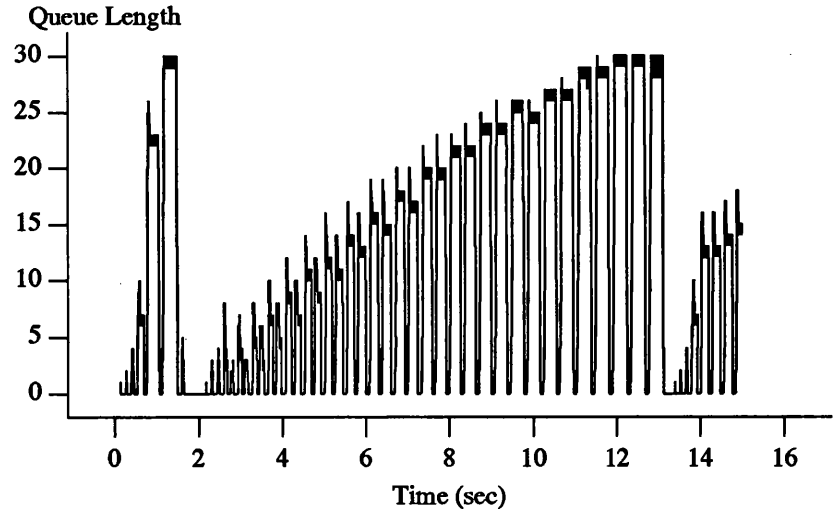
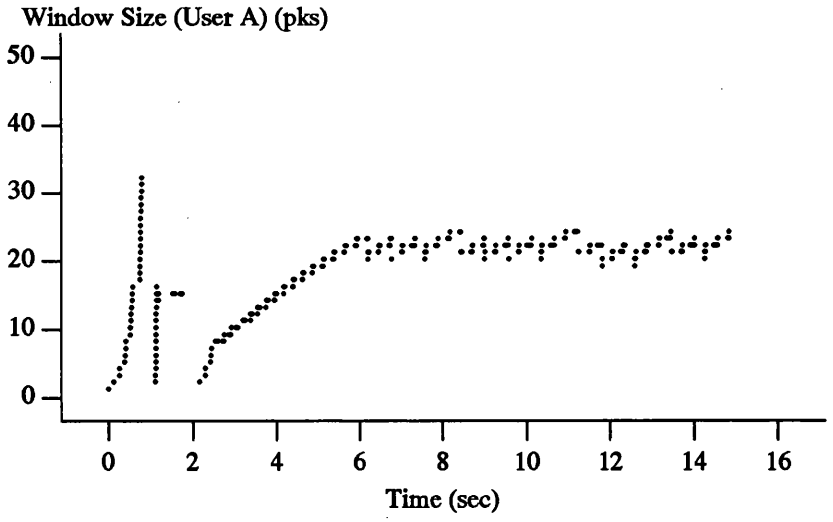
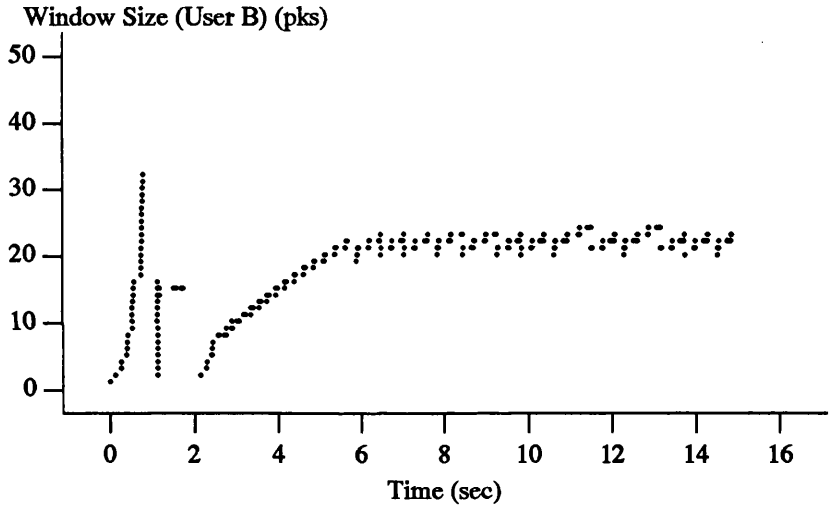


Figure 7.19: Scenario Three (TCP tahoe)





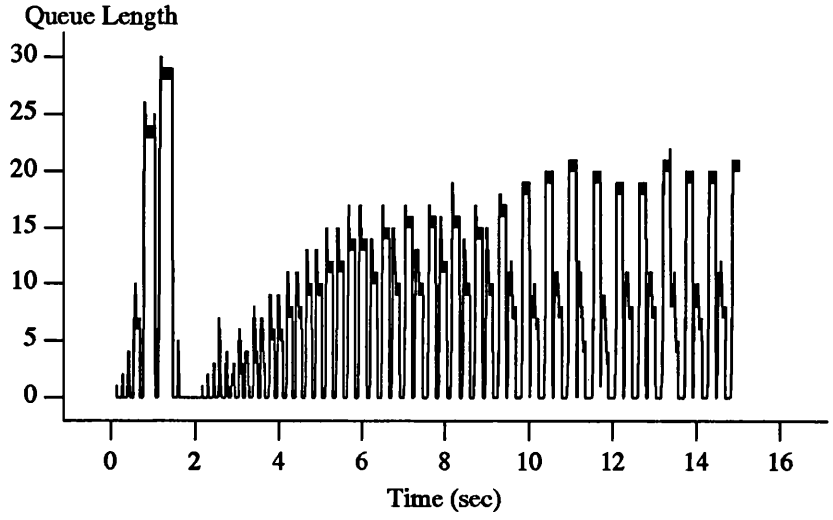


Figure 7.20: Scenario Three (DUAL)

CHAPTER EIGHT

CONCLUSIONS AND FUTURE WORK

8.1. Lessons Learnt

In this dissertation we examined routing and congestion control in a datagram environment. One of the basic problems we have is how to adapt to changing conditions. At an abstract level, the difficulties that routing and congestion control face are extremely similar: inadequate information and delayed feedback. In a network, the only available information is often the state of the aggregate traffic. To make optimal routing and congestion control decisions, however, requires information on each individual flow on the links and bottlenecks. The delay in the feedback information also poses a fundamental limit to any feedback control mechanisms. Any attempts to adjust faster than the information can propagate only result in wild oscillation. The solution that we proposed in this dissertation to solve the problems is *to deal with momentary fluctuations and average steady trends separately*. In the SPF-EE algorithm, we calculate shortest paths based on the average link distance over a long time period and deal with momentary fluctuations with alternate paths. In the STA algorithm, we make rapid adjustments in traffic rates when there are significant traffic changes and leave the momentary fluctuations to buffering and finer adjustment algorithms.

8.2. Summary of the Contributions

This dissertation has made a number of contributions to the areas of routing and congestion control. We summarize the major contributions in this section.

- A detailed discussion on the characteristics and classification of routing algorithms and congestion control schemes were presented. Three aspects were examined in detail: design goals, composition and classification.

- The behavior of shortest-path routing algorithms was examined and six major problems were identified. It has been shown that in a dynamic environment shortest-path routing algorithms often exhibit instability, produce poor routes and consequently degrade the network performance. When the traffic is heavy and changing rapidly, the estimation errors of the link distances increase and the routing algorithm often can not converge to a stable solution and result in wild oscillation.
- A new approach for dynamic routing based on the concept of decentralized decision making was introduced. A new routing algorithm based on the new approach, *Shortest Path First with Emergency Exits (SPF-EE)*, was presented, which eliminates many of the problems that the SPF algorithm exhibits under heavy and dynamic traffic. The idea is to allow local and temporary alteration of routes without global route updating. Simulation has shown that the SPF-EE algorithm has achieved substantial improvements over the SPF algorithm.
- The mechanisms for information feedback in congestion control were investigated. Two different types of approach were identified. Two new binary feedback schemes and one quantitative feedback scheme were proposed.
- A new traffic adjustment algorithm *synchronized traffic adjustment (STA)* was proposed. STA attempts to establish the sharing of the resources only when there are significant traffic changes and to maintain fairness by synchronized adjustment. Three congestion control schemes, based on the STA approach and the information feedback schemes proposed in the previous chapter, were presented. Extensive simulation shows that the STA algorithm eliminates the oscillation problems in the AIMD algorithm and converges more quickly.

8.3. Future Work

A. Enhancements of Routing and Congestion Control in the Internet

Most of the schemes proposed in this dissertation are directly applicable to the Internet. We believe that these schemes can greatly enhance routing and congestion control over the Internet. Due to time and resource constraints, experiments were carried out by simulation. An immediate extension to the current work is to implement these schemes into the Internet protocols and carry out real experiments over the Internet.

B. Decentralized Routing Algorithms

The SPF-EE algorithm is only a very simple application of the concept of decentralized decision making, and is basically an extension to the SPF algorithm. We believe that the concept

of decentralized decision making can be further exploited by designing an algorithm that integrates the computation of shortest paths and alternate paths. The search of alternate paths can also be done by some message passing mechanisms used in the distance-vector algorithms therefore the new algorithm can be a hybrid of link-state and distance-vector algorithms.

C. Routing in ATM Networks

Routing in ATM networks is an issue for further research. Although ATM networks are connection-oriented, many of the routing techniques for datagram networks are still applicable. In an ATM network, operational states are stored inside the networks therefore the network is more vulnerable to failures. On the other hand, however, ATM networks can potentially provide much richer interconnections, hence more redundant paths. Effective routing mechanisms to use alternate paths and to survive failures in ATM networks require further study. In ATM networks there is much more information available to routing which is not generally available in datagram networks. For example, an ATM switch will be informed of the time of the setup and teardown of a connection, and it has access to the state information such as the number of current connections and possibly even their reservation parameters. An interesting study would be to develop a new routing algorithm which can use the additional information effectively and make better routing decisions.

D. Integration of End-System and Network Control

Most of the current work on network congestion control and end-system congestion control are carried out independently. Network control is designed to enforce fair sharing of the bandwidth at the switches. End-system control is to adjust traffic at the source, based on feedback information. An area for further work is to develop an integrated system and establish more effective communications between the end-systems and the network switches, so that the actions by the end-systems and the network switches can be coordinated.

E. Interactions between Routing and Congestion Control

The interactions between routing and congestion control remain an important area for further work. Under congestion routing can disperse traffic via alternate paths while congestion control can reduce traffic at the sources and enforce fairness at the switches. It is important that the information is shared and actions are coordinated. For example, if the congestion control component detects that a particular flow is causing excessive traffic and informs the routing component, it is possible to update the route table in a way that only the route of that particular flow is altered. Such quantitative information can help to avoid the oscillation discussed in chapter

four.

F. An Ultimate Resource Control System

In future high speed networks, admission control will have to be introduced to provide performance guarantees for real-time applications and accounting information. The integration of routing, congestion control and admission control will form an ultimate resource control system for the network which provides integrated traffic control and resource management. In such a system, it is important to have a unified information database and effective mechanisms for message passing between different components. It is also desirable to have distributed control with certain degree of global coordination.

APPENDIX

This Appendix gives a complete description of the SPF-EE algorithm for route computation and packet forwarding. In this description, node x is the node where the algorithm is running. LIST is a list structure for the computation. D_{xz}^{y} denotes the total length of the path from node x to node z via node y. D_{xz} denotes the length of the shortest path from node x to node z that is known at the time of computation. L_{xy} denotes the length of the link between node x and y. T_x and T_x^r represent respectively the routing table and RAP table for node x.

Procedure 1 (route computation for SP)
when node x receives a update and L_{im} is changed by ΔL_{im}
begin
  if $TREE_x$ does not exist **then**
    place D_{xx}^{x} on LIST;
  else
    begin
      if link *im* is in $TREE_x$ **then**
        set $\Delta = \Delta L_{im}$;
      else
        begin
          set $\Delta = D_{xi} + L_{im} + \Delta L_{im} - D_{xm}$;
          if $\Delta \geq 0$ **then**
            stop;
        end
    end
  for each node j in the subtree $TREE_m$ **do**
    set $D_{xj} = D_{xj} + \Delta$;
  for each node j in the subtree $TREE_m$ **do**
    begin
      if $\Delta > 0$ **then**
        begin
          get D_{xj}^{q} so that q $\in N_j$, q $\notin TREE_m$, $D_{xj}^{q} = D_{min}$, $D_{xj}^{q} < D_{xj}$;
          if $D_{xj}^{q} \neq \emptyset$ **then**
            place D_{xj}^{q} on the LIST;
        end
      else
        begin
          get D_{xq}^{j} so that q $\in N_j$, q $\notin TREE_m$, $D_{xq}^{j} = D_{min}$, $D_{xq}^{j} < D_{xq}$;
          if $D_{xq}^{j} \neq \emptyset$ **then**
            place D_{xq}^{j} on the LIST;
        end
  while LIST is nonempty **do**

```
    begin
      get D^l_xn so that D^l_xn ∈ LIST, D^l_xn = D_min;
      remove D^l_xn from LIST;
      place node n on TREE_x so that node l is the predecessor;
      for each node k, k ∈ N_n do
        begin
          if k is in TREE_x and D_xk > D^n_xk then
             begin
               remove node k from TREE_x;
               place D^n_xk on LIST;
             end
          else if D^v_xk (v ≠ n) is in LIST and D_xk > D^n_xk then
             begin
               remove D^v_xk from LIST;
               place D^n_xk in LIST;
             end
          else if D^v_xk (v ≠ n) is not in LIST then
             place D^n_xk in LIST;
        end
    end
      update T_x;
end

Procedure 2 (route computation for AP)
when node x receives a update and L_im is changed by ΔL_im
begin
  for each entry in T^r_x do
    begin
      if source is not node x then
         send MSG(reset, x, z) to source node;
    end
  for each node y, y ∈ N_x do
    begin
      place node y at the root of TREE_y;
      for each node j in the subtree TREE_x do
        begin
          get D^q_yj so that q ∈ N_j, q ∉ TREE_x, D^q_yj = D_min, D^q_yj < D_yj;
          if D^q_yj ≠ ∅ then
             place D^q_yj on the LIST;
        end
    while LIST is nonempty do
      begin
        get D^l_yn so that D^l_yn ∈ LIST, D^l_yn = D_min;
        remove D^l_yn from LIST;
        place node n on TREE_y so that node l is the predecessor;
        for each node k, k ∈ N_n do
          begin
            if k is in TREE_y and D_yk > D^n_yk then
               begin
                 remove node k from TREE_y;
                 place D^n_yk on LIST;
               end
            else if D^v_yk (v ≠ n) is in LIST and D_yk > D^n_yk then
              begin
```

```
                remove D^v_yk from LIST;
                place D^n_yk in LIST;
              end
            else if D^v_yk (v ≠ n) is not in LIST then
              place D^n_yk in LIST;
          end
        end
      end
      update T_x;
end

Procedure 3 (packet forwarding)
when node x receives a packet for node z
begin
  get NS_z from T_x;
  if Q_NS_z < T_s then
    send along SP;
  else
    begin
      get NA_z from T_x or T^r_x;
      if NA_z = ∅ and RAP entry for x, z not marked busy then
        begin
          send MSG(query, x, z) to N_x;
          mark RAP entry for x, z busy;
          send packet along SP;
        end
      else if Q_NA_z < T_a then
        send packet along AP or RAP;
      else
        begin
          send MSG(query, x, z) to N_x;
          send packet along SP;
        end
    end
end

Procedure 4 (search for RAP)
when node w receives MSG(query, x, z)
begin
  for each TREE_n, n ∈ N_w do
    begin
      if destination z is not on TREE_w and TREE_x then
        begin
          record x, z, w and n in T^r_w;
          send MSG(reply, x, z) to node x;
        end
      else
        send MSG(query, x, z) to N_n;
    end
end

Procedure 5 (search for RAP)
when node x receives MSG(reply, x, z)
begin
```

if an entry **for** node x and z does not exist in T_x^r **then**
record RAP path in T_x^r;
end

Procedure 6 (search for RAP)
when node x receives MSG(reset, x, z)
begin
if an entry **for** the path exists in T_x^r **then**
clear this entry;
end

BIBLIOGRAPHY

[As86] H. van As, "Transient analysis of Markovian queueing systems and its application to congestion control modelling," *IEEE Journal on Selected Areas of Communications*, vol. 4, no. 6, Sept. 1986.

[Bara64] P. Baran, "On Distributed Communications, Vols. I-XI," Research Documents, RAND Corporation, Aug. 1964.

[Bert82] D. Bertsekas, "Dynamic Behavior of Shortest Path Routing Algorithms for Communication Networks," *IEEE Transactions on Automatic Control*, vol. AC-27, no. 1, Feb. 1982.

[Bert87] D. Bertsekas and R. Gallager, *Data Networks*, Prentice Hall, 1987.

[Bolo90a] J. Bolot and U. Shankar, "Analysis of a Fluid Approximation to Flow Control Dynamics," Technical Report UMIACS-TR-90-135, Institute for Advanced Computer Studies, University of Maryland, USA, Sept. 1990.

[Bolo90b] J. Bolot and U. Shankar, "Dynamical behavior of rate-based flow control mechanisms," *ACM Computer Communication Review*, vol. 20, no. 2, April 1990.

[Cegr75] T. Cegrell, "A Routing Procedure for the TIADS Message-Swicthing Network," *IEEE Transactions on Communications*, vol. COM-23, no. 6, 1975.

[Chiu89] D. M. Chiu and R. Jain, "Analysis of Increase and Decrease Algorithms for Congestion Avoidance in Computer networks," *Comp. Networks and ISDN System*, vol. 17, pp. 1-14, 1989.

[Clar90] D. D. Clark, "Policy Routing in Internetworks," *Internetworking: Research and Experience*, vol. 1, no. 1, pp. 35-52, Sept. 1990.

[Come90] D. Comer, *Internetworking with TCP/IP: Vol I*, Prentice Hall, 1990.

[Corr79] F. P. Corr and D. H. Neal, "SNA and Emerging International Standards," *IBM System Journal*, vol. 18, no. 2, 1979.

[Coud88] J. P. Coudreuse, W. D. Sincoskie, and J. S. Turner (editors), "Issues on Broadband Packet Switching," *IEEE Journal on Selected Areas of Communications*, vol. 6, no. 9, 1988.

[Crow92] J. Crowcroft, I. Wakeman, and Z. Wang, "Layering Considered Harmful," *IEEE Networks*, Jan. 1992.

[Cyer87] R. Cyert and M. DeGroot, *Bayesian Analysis and Uncertainty in Economic Theory,* Chapman and Hall, 1987.

[Deme89] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *Proc. of ACM SIGCOMM'89*, pp. 1-12, Austin USA, Sept. 1989.

[Dijk59] E. W. Dijkstra, "A Note on Two Problems in Connection with Graphs," *Numerical Mathematics*, vol. 1, pp. 269-271, 1959.

[Dijk80] E. W. Dijkstra and C. S. Scholten, "Termination Detection for Diffusing Computations," *Information Processing Letters*, vol. 11, no. 1, 1980.

[Folt83] H. C. Folts and R. desJardins (editors), "Special Iusse on Open Systems Interconnection (OSI)," *Proceedings of IEEE*, vol. 71, no. 2, Dec. 1983.

[Ford62] L. R. Ford and D. R. Fulkerson, *Flows in Networks,* Princeton University Press, 1962.

[Garc86] J. Garcia-Luna-Aceves, "A Fail-Safe Routing Algorithm for Multihop Packet-Radio Network," *Proc. of INFOCOM'86*, pp. 434-443, 1986.

[Garc88] J. Garcia-Luna-Aceves, "A Distributed, Loop-Free, Shortest-Path Routing Algorithm," *Proc. of INFOCOM'88*, pp. 1125-1137, 1988.

[Garc89b] J. Garcia-Luna-Aceves, "A Minimum-Hop Routing Algorithm Based on Distributed Information," *Computer Networks and ISDN systems*, vol. 16, no. 5, pp. 367-382, May 1989.

[Garc89a] J. Garcia-Luua-Aceves, "A Unified Approach for Loop-Free Routing Using Link States or Distance Vectors," *ACM SIGCOMM'89*, pp. 212-223, 1989.

[Gerl80] M. Gerla and L. Kleinrock, "Flow Control: A Comparative Survey," *IEEE Transactions on Communications*, vol. COM-28, no. 4, April 1980.

[Hedr89] C. L. Hedrick, "An Introduction to IGRP," Technical Report, RUTGERS, Centre for Computers and Information Services, The State University of New Jersey, Oct. 1989.

[Heyb89] A. Heybey, *The Network Simulator,* MIT, 1989.

[Jaco88] V. Jacobson, "Congestion Avoidance and Control," *Proc. of ACM SIGCOMM'88*, pp. 314-329, Stanford, USA, Aug. 1988.

[Jaco90] V. Jacobson, R. Braden, and L. Zhang, "TCP extension for high-speed paths," RFC 1185, 1990.

[Jaff82] J. M. Jaffe and F. H. Moss, "A Responsive Distributed Routing Algorithm for Computer Networks," *IEEE Transactions on Communications*, vol. COM-30, no. 7, pp. 1758-1762, 1982.

[Jain86] R. Jain, "A Timeout-Based Congestion Control Scheme for Window Flow-Controlled Networks," *IEEE Journal on Selected Areas of Communications*, vol. SAC-4, no. 7, pp. 1162-1167, Oct. 1986.

[Jain89] R. Jain, "A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks," *Computer Communication Review*, vol. 19, no. 5, pp. 56-71, Oct. 1989.

[Kesh91a] S. Keshav, A. K. Agrawala, and S. Singh, "Design and Analysis of a Flow Control Algorithm for a Network of Rate Allocating Servers," *Protocols for High Speed Networks II*, Elsevier Science Publishers, Holland, April 1991.

[Kesh91b] S. Keshav, "Congestion Control in Computer Networks," *PhD Thesis*, University of California at Berkeley, August 1991.

[Kesh91c] S. Keshav, "A Control-Theoretic Approach to Flow Control," *Proc. of ACM SIGCOMM'91*, Sept. 1991.

[Khan89] A. Khanna and J. Zinky, "The Revised ARPANET Routing Metric," *Proc. of ACM SIGCOMM'89*, pp. 45-56, Sept. 1989.

[Klei64] L. Kleinrock, *Communication Nets: Stochastic Message Flow and Delay,* McGraw-Hill, 1964.

[Klei76] L. Kleinrock, *Queueing Systems: Volume II,* John Wiley & Sons, 1976.

[Luce57] R. Luce and H. Raiffa, *Games and Decisions,* John Wiley & Sons, Inc., 1957.

[Maji79] J. C. Majithia, "Experiments in Congestion Control Techniques," *Proc. of International Symposium on Flow Control in Computer Networks*, Feb. 1979.

[Mart88] D. Martin, *The Network Simulator User Manual*, MIT, 1988.

[McQu74] J. M. McQuillan, "Adaptive Routing Algorithms for Distributed Computer Networks," PhD Thesis, Harvard University, 1974.

[McQu77] J. M. McQuillan and D. C. Walden, "The ARPA Network Design Decision," *Computer Networks*, vol. 1, pp. 243-389, 1977.

[McQu79] J. M. McQuillan, "Interactions Between Routing and Congestion Control in Computer Networks," *Proc. of International Symposium on Flow Control in Computer Networks*, pp. 63-75, North-Holland, Feb. 1979.

[McQu80] J. M. McQuillan and I. Richer, "The New Routing Algorithm for the ARPANET," *IEEE Transactions on Communications*, vol. COM-28, no. 5, pp. 711-719, May 1980.

[Merl79] P. M. Merlin and A. Segall, "A Failsafe Distributed Routing Protocol," *IEEE Transactions on Communications*, vol. COM-27, no. 9, pp. 1280-1287, 1979.

[Mitr90] D. Mitra and J. Seery, "Dynamic Adaptive Window for High Speed Data Networks: Theory and Simulations," *Proc. of ACM SIGCOMM'90*, Sept. 1990.

[Moy89] J. Moy, "The OSPF Specification," RFC 1131, SRI International, USA, Oct. 1989.

[Nagl87] J. Nagle, "On Packet Switches with Infinite Storage," *IEEE Transactions on Communications*, vol. COM-35, no. 4, pp. 435-438, April 1987.

[Newe71] G. F. Newell, *Applications of Queueing Theory*, Chapman and Hall, London, 1971.

[Orga89] International Standards Organization, "Intra-Domain IS-IS Routing Protocol," *ISO/IEC JTC1/SC6 WG2 N323*, Sept. 1989.

[Post81a] J. Postel, "Internet Protocol (IP)," RFC 791, SRI International, USA, Sept. 1981.

[Post81b] J. Postel, "Transmission Control Protocol (TCP)," RFC 793, SRI International, USA, 1981.

[Pric77] W. L. Price, "Data Network Simulation Experiments at the National Physical Laboratory," *Computer Networks*, vol. 1, no. 1, 1977.

[Prue87] W. Prue and J. Postel, "Something a Host Could Do with Source Quench: The Source Quench Introduced Delay (SQuID)," RFC 1016, SRI International, USA,

July 1987.

[Quar90] J. Quarterman, *The Matrix,* Digital Press, 1990.

[Rama87] K. K. Ramakrishnan, R. Jain, and D.M.Chiu, "Congestion Avoidance in Computer Networks with a Connectionless Network Layer, Part IV: A Selective Binary Feedback Scheme for General Topologies," DEC Technical Report TR-510, 1987.

[Rama88] K. K. Ramakrishnan and R. Jain, "An Explicit Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer," *Proc. of ACM SIGCOMM'88,* pp. 303-313, Stanford, USA, Aug. 1988.

[Ride76] K. L. Rider, "A Simple Approximation to the Average Queue Size in the Time-Dependent M/M/1 Queue," *Journal of the ACM,* vol. 23, no. 2, April 1976.

[Robe78] L. G. Roberts, "The Evolution of Packet Switching," *Proceedings of the IEEE,* vol. 66, no. 11, pp. 1307-1313, 1978.

[Rose81] E. Rosen, "Vulnerability of Network Control Protocols: an Example," RFC 789, SRI International, USA, 1981.

[Saat61] T. Saaty, *Elements of Queueing Theory,* McGraw-Hill, 1961.

[Schw80] M. Schwart and T. Stern, "Routing Techniques Used in Computer Communication Networks," *IEEE Transactions on Communications,* vol. COM-28, no. 4, pp. 539-552, 1980.

[Seeg86] J. Seeger and A. Khanna, "Reducing Routing Overhead in a Growing DDN," *Proc. of MILCOM'86,* pp. 15.3.1-15.3.13, Oct. 1986.

[Shen90] S. Shenker, L. Zhang, and D. Clark, "Some Observations on the Dynamics of a Congestion Control Algorithm," *ACM Computer Communication Review,* vol. 20, no. 5, Oct. 1990.

[Shin87] K. G. Shin and M-S. Chen, "Performance Analysis of Distributed Routing Strategies Free of Ping-Pong-Type Looping," *IEEE Transactions on Communication,* vol. COM-36, no. 2, pp. 129-137, 1987.

[Sing90] S. Singh, A. K. Agrawala, and S. Keshav, "Deterministic Analysis of Flow and Congestion Control Policies in Virtual Circuits," Technical Report 2490, Computer Science Department, University of Maryland, USA, June 1990.

[Sore80] H. Sorenson, *Parameter Estimation,* Marcel Dekker, Inc., 1980.

[Ster80] T. E. Stern, "An Improved Algorithm for Distributed Computer Networks," *Proc. of IEEE Symposium on Circuits and Systems*, pp. 2-6, 1980.

[Taji77] W. D. Tajibnapis, "A Correctness Proof of a Topology Information Maintenance Protocol for a Distributed Computer Network," *Communication of the ACM*, vol. 20, pp. 477-485, 1977.

[Taka62] L. Takacs, *Introduction to the Theory of Queues,* Oxford University Press, 1962.

[Turn86] J. S. Turner, "New Directions in Communications (or Which Way to the Information Age?)," *IEEE Communications*, vol. 24, no. 10, pp. 8-15, Oct. 1986.

[Wacl90] J. G. Waclawsky and A. K. Agrawala, "Dynamic Behavior of Data Flow within Virtual Circuits," Technical Report 2250, Computer Science Department, University of Maryland, USA, May 1990.

[Wang89] Z. Wang, "Model of Network Faults," *Proceedings of the First International Symposium on Integrated Network Management*, Boston, USA, 1989.

[Wang90] Z. Wang and J. Crowcroft, "Shortest Path First with Emergency Exits," *Proceedings of ACM SIGCOMM'90*, Philadelphia, USA, 1990.

[Wang91] Z. Wang and J. Crowcroft, "A New Congestion Control Scheme: Slow Start and Search (Tri-S)," *ACM Computer Communication Review*, vol. 21, no. 1, Jan. 1991.

[Wang92a] Z. Wang and J. Crowcroft, "Analysis of Shortest-Path Routing Algorithms in a Dynamic Environment," *ACM Computer Communication Review*, vol. 22, no. 2, April 1992.

[Wang92b] Z. Wang and J. Crowcroft, "A Fluid Model Approximation to Quantitative Information Feedback in Congestion Control," *Proc. of IEEE 7th International Phoenix Conference on Computers and Communications*, Arizona, USA, 1992.

[Wang92c] Z. Wang and J. Crowcroft, "Eliminating Periodic Packet Losses in the 4.3-Tahoe BSD TCP Congestion Control Algorithm," *ACM Computer Communication Review*, Jan. 1992.

[Wang92d] Z. Wang, J. Crowcroft, and I. Wakeman, "A Simple TCP Extension for High-Speed Paths," *Submitted to ACM Computer Communication Review*, Jan. 1992.

[Weck80] S. Wecker, "The Digital Network Architecure," *IEEE Transactions on Communications*, vol. Com-28, 1980.

[Xero85] Xerox, "Xerox Network System Architecture," *Xerox Corporation Document 068504*, 1985.

[Zhan91a] H. Zhang and S. Keshav, "Comparison of Rate-Based Service Disciplines," *Proc. of ACM SIGCOMM'91*, Sept. 1991.

[Zhan89] L. Zhang, "A New Architecture for Packet Switching Network Protocols," *PhD Thesis*, MIT, 1989.

[Zhan90] L. Zhang, "VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks," *Proc. of ACM SIGCOMM'90*, Philadelphia, USA, Sept. 1990.

[Zhan91b] L. Zhang, S. Shenker, and D. Clark, "Observation on the Dynamics of a Congestion Control Algorithm: the effects of Two-Way Traffic," *Proc. of ACM SIGCOMM'91*, 1991.