A STUDY OF QUALITY OF SERVICE COMMUNICATION FOR HIGH-SPEED PACKET-SWITCHING COMPUTER SUB-NETWORKS

By

Zhenqian Cui

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computer Science
in the Department of Computer Science

Mississippi State, Mississippi

May 1999

Name: Zhenqian Cui

Date of Degree: May 13, 1999

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Anthony Skjellum

Title of Study: A STUDY OF QUALITY OF SERVICE COMMUNICATION FOR HIGH-SPEED PACKET-SWITCHING COMPUTER SUB-NETWORKS

Pages in Study: 80

Candidate for Degree of Master of Science

In this thesis, we analyze various factors that affect quality of service (QoS) communication in high-speed, packet-switching sub-networks. We hypothesize that sub-network-wide bandwidth reservation and guaranteed CPU processing power at endpoint systems for handling data traffic are indispensable to achieving hard end-to-end quality of service. Different bandwidth reservation strategies, traffic characterization schemes, and scheduling algorithms affect the network resources and CPU usage as well as the extent that QoS can be achieved. In order to analyze those factors, we design and implement a communication layer. Our experimental analysis supports our research hypothesis. The Resource ReSerVation Protocol (RSVP) is designed to realize resource reservation. Our analysis of RSVP shows that using RSVP solely is insufficient to provide hard end-to-end quality of service in a high-speed sub-network. Analysis of the IEEE 802.1p protocol also supports the research hypothesis.

ACKNOWLEDGEMENTS

I am in debt to my advisor Dr. Anthony Skjellum for providing me valuable guidance and assistance throughout my Master's program. His understanding and encouragement were indispensable for the completion of this thesis. I am also grateful to Dr. Bradley Carter and Dr. Raghu Machiraju for their valuable suggestions and feedback.

I would like to acknowledge that the Defense Advanced Research Project Agency (DARPA) provided partial funding through the United Air Force Research Laboratory, Rome, New York, under contracts F30602-95-1-0036 ("MPI/RT and PacketWay" study) and F30602-96-1-0329 ("Ubiquitous Realtime NOWs" study) for conducting this research. I also acknowledge National Aeronautics and Space Administration (NASA) for providing additional funding under contract NAG2-1211 for conducting this research ("Two-level Middleware" study).

I am thankful to my colleagues in the High Performance Computing Laboratory (HPCL). The author is especially grateful to Shane Herbert and Jin Li for valuable discussions and suggestions. I am also grateful to Gerhard Lehnerer for helping me set up the testbed for the experiments.

Finally, I express special appreciation to my wife, Shanshan Lin, and my parents for their persistent support. Without their support and encouragement, it would have been impossible to finish this thesis.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

TABLE Page

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

With the development of high-speed networking technology, computer networks, including local-area networks (LANs), wide-area networks (WANs) and the Internet, are extending their traditional roles of carrying computer data. They are being used for Internet telephony, multimedia applications such as conferencing and video on demand, distributed simulations, and other real-time applications. LANs are even used for distributed real-time process control and computing as a cost-effective approach. Differing from traditional data transfer, these new classes of high-speed network applications (video, audio, real-time process control, and others) are delay sensitive. The usefulness of data depends not only on the correctness of received data, but also the time that data are received. In other words, these new classes of applications require networks to provide guaranteed services or quality of service (QoS). Quality of service can be defined by a set of parameters and reflects a user's expectation about the underlying network's behavior. Traditionally, distinct services are provided by different kinds of networks. Voice services are provided by telephone networks, video services are provided by cable networks, and data transfer services are provided by computer networks. A single network providing different services is called an integrated-services network.

Providing integrated services over packet-switching networks is attractive for two key reasons. First, the infrastructure is often already in place and network bandwidth is increasing rapidly. Packet-switching networks represent the latest network technology. High bandwidth makes feasible that switching networks provide such integrated services, since QoS-oriented communications usually need a lot of peak bandwidth for handling bursty traffic. Second, an integrated service network seems more economical and easier to manage than separate datagram networks and real-time networks, working in parallel.

Unlike circuit-switching networks (such as telephone networks), computer networks are essentially datagram-based networks. They are designed to provide best-effort service, which is sufficient for the data transfer service they provide. In datagram networks, each packet is routed independently across shared networks and is possibly reassembled with other datagrams at the receiving side to construct a complete message so as to achieve the maximum usage of network resources. There is no dependable way to know in advance how much time it will take for a packet to be transferred from the sending endpoint to the receiving endpoint: The transfer times vary significantly because of the dynamically changing network loads. In other words, delay bounds are broad. Consequently, providing QoS in a LAN or WAN emerges as a new challenge.

There are many approaches proposed in the literature to meet this QoS challenge. Those approaches can be divided into two categories. One class of methods includes modification of the current link-layer protocol (such as Ethernet or token ring protocols) in order to make it appropriate for real-time applications. These methods are usually

restricted to a single LAN, or even a network segment. The IEEE802.1p protocol is such a protocol, designed to provide bandwidth reservation in an Ethernet network. As another example, Asynchronous Transfer Mode (ATM) is a link-layer protocol that is designed to provide connection-based quality of service. Another class of methods works with WANs (including the Internet) in order to provide QoS, despite the underlying complexity and heterogeneity of the constituents of the network. This class of methods supposes that the underlying sub-networks provide delay-bound guarantees. However, in the Internet environment, parts are often heavily loaded, which results in congestion, with consequent indefinite packet delay. Several protocols are proposed for time-sensitive applications over the Internet such as resource ReSerVation (RSVP) protocol (Braden et al. 1997). But, numerous problems still need to be resolved before RSVP can be deployed in an open environment. Specifically, policy control is an ongoing research topic in the RSVP forum.

In this thesis, our research focus is on how to provide quality of service in a closed sub-network environment and to analyze various factors affecting end-to-end QOS. A sub-network is a homogeneous part of a LAN (or possibly a whole LAN) that connects a cluster of workstations through switches. This kind of switching network is typically used for real-time process control and real-time distributed computing (Mizunuma, Shen and Takegaki 1996). A closed sub-network means that the users have complete control over all network elements in this sub-network and that there is no traffic interference from outside. Typical network elements include switches, hosts, and physical links. Because the network resources are shared by all applications running on hosts connected

to networks, any real-time application that tries to use network resources and is not controlled by the system communication middleware would interfere with other real-time applications. Hence, such applications would break QoS guarantees that were granted to those pre-existing, properly admitted applications. Consequently, in a switching sub-network, not only the link layer is required to provide bound packet transfer delay, but also a resource reservation mechanism is necessary to assure compliance.

Given these issues, the research hypothesis for this work is that sub-network-wide bandwidth reservation and guaranteed host CPU processing power for handling data traffic are both indispensable to achieving hard end-to-end quality of service. The link layer must first provide bounded delay, otherwise no bounded end-to-end delay is possible for message transfer. Different bandwidth reservation strategies, traffic characterization schemes, and scheduling algorithms can affect the network resources and CPU usage as well as the extent that QoS can be achieved, and are also needed.

Our research strategy is empirical. We first offer an analysis on the IEEE 802.1p protocol to test our hypothesis. The IEEE 802.1p protocol is a typical link-layer bandwidth reservation protocol. Then we offer an analysis on RSVP to test the research hypothesis. Finally we create our own experimental communication layer middleware to investigate QoS in high-speed packet-switching sub-networks and to test the research hypothesis further. In this middleware, we introduce a sub-network-wide and topology-based resource reservation mechanism. We show that this resource reservation mechanism is more efficient compared to RSVP. We execute test cases on high-

performance Sun Solaris workstations that are connected together using gigabit-per-second Ethernet switches. These switches provide recent QoS-oriented extensions to Ethernet.

The remainder of this thesis is organized as follows. Chapter II reviews the relevant literature in real-time communication over packet-switching networks and discusses various mechanisms and approaches needed to efficiently use a datagram network for real-time communication. Chapter III presents link-layer protocols and resource reservation protocols for quality of service and offers an experimental analysis on the IEEE 802.1p protocol and on the RSVP protocol, respectively. In Chapter IV, we present the design and implementation of our experimental communication layer, and then, in Chapter V, we offer an analysis based on testing results. In Chapter VI, we conclude the research and present lessons learned and possible future work.

CHAPTER II

LITERATURE REVIEW

The purpose of this chapter is to review the relevant literature in the research of quality of service communication in packet-switching networks. Various protocols and algorithms designed for providing quality of service communication are reviewed.

Key Terms Related to This Work

We define the following key terms to simplify the exposition that follows. These terms are frequently referred to throughout this thesis.

Sub-network: A sub-network is a homogeneous part of a LAN (or possibly a whole LAN) that connects a cluster of workstations through switches. A closed sub-network means that the users have complete control over all network elements in this sub-network and there is no traffic interference from outside network elements or systems.

Quality of Service (QoS): QoS is a description of the expected or required service of a network by a certain application. Bandwidth, maximum end-to-end packet transfer delay, maximum end-to-end packet transfer delay jitter, and packet loss rate are important parameters used to measure or quantify QoS (Banerjea et al. 1996). "Hard QoS" means absolute guarantees from the underlying network. "Soft QoS" means statistical guarantees from the underlying networks.

Admission Control: Admission control is a mechanism to test whether the underlying networks can accommodate the requested QoS. An admission test is equivalent to admission control.

Jitter: Jitter is the variance of end-to-end packet transfer delay. Smooth jitter means that the variance of packet transfer delay is small. Non-smooth jitter means that the end-to-end packet transfer delay change dramatically.

Reliable service: Reliable service is a kind of service in which networks deliver data packets without error and with bounded delay. If a packet is lost for some reason, the protocol will retransmit the packet until it is received by the receive endpoint, with fault notification after many retries.

Best-effort service: Best-effort service means that the network promises not to delay or discard packets intentionally and does its best to forward packets to the next hop or destination. However, packets may be dropped for reasons of congestion or error.

Real-Time Communication in Packet-Switching Networks

Much research has been done during the past decade toward achieving real-time communication in packet-switching networks. Traditionally, datagram networks only provide best-effort services in which there are no guarantees as to whether a packet will be delivered reliably to the destination and when it will arrive at the destination. Reliable data transfer is achieved by transport-layer protocol such as TCP. Many studies show that without link-layer real-time protocol support, it is difficult to provide hard end-to-end quality of service (QoS) in a packet-switching network. This is so

because the link layer may add indefinite end-to-end packet transfer delay, or because the delay variance of two consecutive packets may be larger than the expected value (Kurose 1993). Hard end-to-end QoS means that underlying networks provide absolute guarantees on requested QoS. Correspondingly, soft QoS means statistical service guarantees from networks.

Higher layer protocols and resources management are necessary to provide QoS in wide-area networks (Clark, Shenker, and Zhang 1992). Some link-layer protocols for real-time communication were proposed, such as sub-network bandwidth reservation protocol on Ethernet, real-time Media Access Control protocols, ATM, and others. Various factors can cause end-to-end packet-transfer delay. Figure 2.1 shows possible delay that a packet may experience from the sending endpoint to the receiving endpoint.

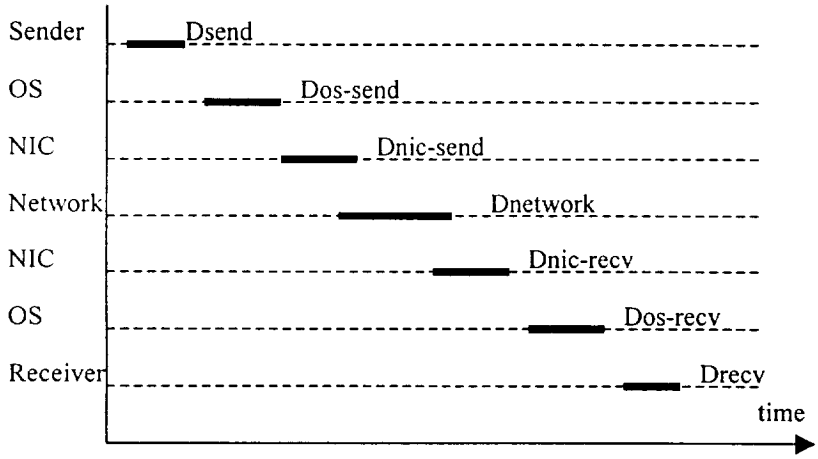


Figure 2.1. End-to-End Packet Transfer Delay Distribution

When the sender generates packets in real time and passes them to the operating system, a packet may experience indefinite delay inside the operating system because of

process or thread scheduling. When the packet arrives at the network interface card (NIC), depending on the underlying network, it may also experience indefinite delay. However, in a switching network the delay caused by the NIC is bounded, but traffic from different sources may conflict inside switches, which forces the upper layer protocol to retransmit data, again resulting indefinite delay. Though in most situations, those indefinite delays are still bounded, the delay bound is too loose to be useful for real-time applications.

From the above discussion, we can see that a fundamental problem for real-time communication in a packet-switching network is to enforce a real-time service model on all network elements and hosts. Such a real-time service model can be achieved though reserving resources and scheduling packets.

There are two basic goals in providing QoS in packet-switching networks. One goal is to effectively use underlying resources, which include network bandwidth, CPU processing power, buffer space, etc. Another goal is that a newly admitted connection should not affect already established connections. The difficulty is that the datagram networks in essence are packet-based and not connection-based. From a user's point of view, those connections should be independent of each other. So, any complete approach intended to provide QoS in packet-switching networks needs to deal with the following three issues (Ferrari and Verma 1989): 1) QoS and traffic specification models; 2) the admission control and resource reservation models; and 3) packet scheduling strategies or service disciplines. During the past decade, many packet

scheduling algorithms, and traffic models have been proposed. In the following sections, we review these issues.

Quality of Service Models

Quality of service (QoS) is a description of user-expected service on a network. QoS parameters usually include bandwidth, maximum end-to-end packet transfer delay, maximum end-to-end packet transfer delay jitter, and packet loss rate (Banerjea et al. 1996). The variations of transit delay are called jitter. Jitter is used to measure the delay variance between two consecutive packets. Delay and delay jitter are based on a given packet size. Of the QoS parameters, the end-to-end packet transfer delay bound is the most important for real-time communication since continuous media applications and real-time control applications require bounded packet transfer delay. They also require that packet transfer delay jitter should be bounded.

Packet loss may be caused by the physical link (data corruption), or because packets are discarded intentionally by packet-scheduling algorithms in the cases of delay bound violation, delay jitter violation, or resource exhaustion. Reliable, guaranteed services provide both zero packet loss rate and bound end-to-end delay while statistical services provide only statistical guarantees on delay and jitter and allow a non-zero loss rate (Banerjea et al. 1996). Real-time process control and real-time distributed computing require networks to provide reliable and guaranteed service. Multimedia applications usually need only statistical services because people can tolerate data loss to some degree without noticing it. One advantage of statistical services is that it can greatly increase the network usage since it does not need to reserve resource in terms of peak

rate. Guaranteed service must consider the worst-case delay that a packet may experience across the networks.

Both RSVP and the Tenet protocol suite provides statistical and guaranteed services (Braden, Zhang, Berson, Herzog, and Jamin. 1997; Ferrari and Verma 1989). The real-time Message Passing Interface (MPI/RT) (MPI/RT Forum 1998) strives to provide a standard API for message passing with QoS in a distributed computing environment. For MPI/RT, message passing is reliable and deterministic. MPI/RT also provides best-effort service as a default option.

Traffic Characterization Models

Traffic characterization is represented by a set of parameters that specify the data generation characteristics for a source. The characterization is specified in terms of bounds on data volumes. Based on traffic characterization and QoS requirements, admission control can reserve resources to provide the required QoS. Traffic parameters can be viewed as QoS parameters since they define the lower bounds on instantaneous and average throughput that the network is being requested to provide (Banerjea et al. 1996). There are many other traffic specification models proposed in the literature as well. Several of these models are discussed here.

[Xmin, Smax] (Golestani 1990) is a simple model intended for smooth traffic sources. A smooth traffic source means that the variance of inter-arrival time is zero or small. A connection satisfies this model if the minimum inter-arrival time between two consecutive packets is always equal to or longer than Xmin, and largest packet size is Smax. The peak rate is equal to Smax/Xmin. Using this model will result in over-

reserved resources for statistical service since statistical service does not need to reserve resource in terms of its peak rate.

[Xmin, Xave, I, Smax] was proposed by Zhang and Ferrari (1993) and was used in the Tenet real-time protocol suite (Banerjea et al. 1996). It is suitable for describing non-smooth traffic. Non-smooth traffic means that inter-arrival time between two consecutive packets changes dramatically. The limitation of this model is that it is hard to obtain those parameters except for some well-known sources (such as MPEG streams). This model states that average inter-arrival time of two consecutive packets during any interval of length I must be larger or equal to Xave. The average packet arrival rate is Smax/Xave. For statistical service, the system need only reserve resources in terms of average rate; therefore, the usage of network resources increases. The $[\sigma, \rho]$ model proposed by Cruz (1991) has similar capabilities, but does not specify minimum inter-arrival time, so it is only for statistical service. The σ and ρ parameters are the maximum burst size and the long-term average rate of the source traffic, respectively. During any interval of length t, the number of bits generated by the connection in an interval is less than $\sigma + \rho$ *t.

The Leaky-bucket model is a traffic-conformance model (Turner 1986). It uses a peak rate p and an average rate r to describe traffic, and a third parameter b, token buffer size (or the bucket depth), in order to conform the traffic. Tokens are generated at a fixed rate as long as the token buffer is not full. When a packet arrives from the source, it is released into the network only if there is at least one token in the token buffer, otherwise it will be discarded. This model enforces token arrival rate on the

input stream. Token generation rate should be greater than the packet arrival rate r and less than peak rate p for stability reasons. In terms of this model, during any interval of length t, the number of bits generated will be less than $b + p*t$. The RSVP protocol uses this model as its traffic description model (Braden et al. 1997).

Some more complex traffic models have also been proposed for characterizing the traffic more accurately such as the Deterministic Bounding Interval-Dependent (D-BIND) model (Wrege et al. 1996), which uses multiple bounding average rates, each over a different interval. Its precise characterization of traffic would improve resource usage, but it is hard to use in practice since multiple bounding average rates must be obtained in advance through experimentation.

Packet-Scheduling Algorithms

Packet-scheduling algorithms are also called the queuing mechanisms at a switch or a host. The purpose of such algorithms is to schedule incoming packets for transmission. Figure 2.2 shows basic scheduling problem (Aras et al. 1994). The simplest queuing algorithm is First Come First Served (FCFS). Obviously this algorithm cannot classify and prioritize traffic and is not suitable for real-time communication in packet-switching networks. It can provide only best-effort service. Some widely recognized packet scheduling algorithms are the Weighted Fair Queuing (WFQ) (Parekh 1992), Early Deadline First (EDF) (Ferrari and Verma 1989) and Class-Based Queuing (CBQ) (Floyd and Jacobson 1995) algorithms.

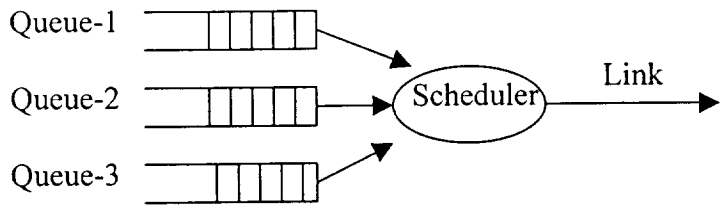


Figure 2.2. Basic Scheduling Problem

There are some basic requirements for any scheduling algorithm. These requirements include separation of connections, efficient resource utilization, fairness among connections, simplicity, and scalability (Hyman, Lazar and Pacifici 1991). Separation of connections means that a misbehaving connection should not affect the well-behaving connections. A connection is misbehaving if it sends data at a rate greater than its negotiated rate. Efficient resource utilization requires that the scheduler be able to allocate resources in terms of QoS requirement and not waste resources. The greater the utilization, the larger the number of connections that can be admitted under the same conditions. Fairness means that if a connection uses less than its negotiated rate, the unused quantity should be evenly divided among the other connections in some way that does not favor any connection over another. Simplicity requires that the scheduler should not consume too much CPU resources. Otherwise the scheduler itself will introduce delay overhead to packet transfer. Simplicity reduces the residence time of a packet at switches and hosts. Scalability means that the scheduler should be able to scale well to cases with large numbers of connections since a physical link may have thousands of logical connections to serve at a network node.

In a real-time distributed computing environment, isolation, simplicity, and scalability are especially important. As for network utilization, there is a tradeoff in complexity and utilization. Utilization represents the actual resources that can be used by the user's applications. The simplicity of a scheduling algorithm can increase the robustness of a system perhaps at the cost of low resource utilization. For hard real-time distributed application, robustness has the most importance.

Weighted Fair Queuing (WFQ) is a packet-by-packet transmission scheme which closely approximates Fluid-Flow Fair Queuing (FFQ). FFQ is a hypothetical, perfect scheduling algorithm in the sense that a packet is infinitely divisible. The implementation of WFQ is based on following equation (Demers, Keshav and Shenker 1989):

$$F_k^i = \max(F_k^{i-1}, v(a_k^i)) + \frac{S_k^i}{\phi_k} \tag{2.1}$$

where F_k^i is the virtual completion time for ith packet on connection k. The a_k^i's denote the arrival time of the ith packet on connection k, S_k^i is the ith packet size on connection k, ϕ_k is the bandwidth assigned to the connection k. The parameter □□is the virtual time function, which is always increasing. Whenever the scheduler is ready to transmit its next packet, it picks up the packet with minimum F value among all packets backlogged for service. The WFQ algorithm uses the maximum burst size and a long-term average rate as source traffic parameters. So, evidently, the leaky bucket model can be used for the WFQ implementation. The WFQ algorithm gives an end-to-

end delay bound if the traffic conforms the negotiated rate. Otherwise the WFQ algorithm cannot provide any guarantee on delay and delay jitter.

The delay-based Early Deadline First (EDF) was proposed by Ferrari and Verma (1989) and is based on the traffic description [Xmin, Xave, I, Smax]. Differing from the WFQ algorithm that requires maintaining a queue for each real-time connection, the EDF algorithm need only maintain three queues. The first queue contains traffic that requires deterministic guarantees on delay. The second contains traffic that requires statistical guarantees. The third contains other traffic without any real-time requirements. When the scheduler needs to send a packet, it compares the ending time of the packet in the statistical queue with the beginning time (i.e., the deadline minus the service time) of the packet in the deterministic queue. If the latter is lower than the former, the next packet is taken from the deterministic queue. Otherwise, the same comparison is made between the no-guarantee queue and the statistical queue, and a decision is made between the two. The EDF algorithm can provide guarantees for bandwidth and end-to-end delay bounds as well as statistical guarantees in which loss-rate resulting from missed deadlines or buffer overflow can be bounded probabilistically. The buffer space needed at each node is also bounded. Admission will reserve a bandwidth of $\frac{1}{X^k_{\min}}$ to each channel k at every node n along its path. However, EDF's admission method is complex.

The admission control performs two tests at each node: a node saturation test and a scheduler saturation test. The node saturation test tests whether the node has sufficient

processing or transmission capacity. The purpose of the scheduler saturation test is to look for the minimum local delay bound for a new channel that does not saturate the scheduler, even in the worst case (Cilingiroglu, Lee and Agrawala 1997). A new channel is accepted if node saturation test succeeds at each node and the sum of local delay bounds is less than or equal to the end-to-end delay bound. EDF is based on the observation that an arriving packet does not need to be sent out immediately as long as it can satisfies the local delay bound.

The class-based queuing (CBQ) algorithm is a more recent technology. It classifies packets in the same way as simple, priority-based algorithms and puts packets into different queues. However, the scheduler serves the queues in a round-robin order. The number of packets that can be removed from a queue on each pass is configured during the admission test in terms of the required QoS. This feature ensures that no class achieves more than a given proportion. Coupled with a timer, the CBQ algorithm can be used to ensure that each class will obtain a certain percentage of bandwidth under any circumstance. Inside each class, CBQ still uses First Come First Serve (FCFS). But, between classes, the CBQ algorithm enforces a certain-degree of fairness. Because of the simplicity and effectiveness of the CBQ algorithm, we use this algorithm for our communication layer implementation.

Many other complex queuing algorithms are also proposed, such as rate control static priority (RCSP) (Zhang and Ferrari 1993), jitter-based Early Deadline First (Ferrari 1992) and the virtual clock algorithm. The virtual clock algorithm gives exactly

the same results as WFQ, but it was derived from Time Division Multiplexing (TDM) (Zhang 1991).

Admission-Control Protocols for Quality of Service Communication

An admission-control protocol is used for establishing a point-to-point connection or multicast connection. Its purpose is to test whether each network element along the path can meet the requested QoS. Several complete protocols have been proposed for admission control. Each is intended to provide a complete method for real-time communication in a wide-area network. The resource ReSerVation Protocol (RSVP) was first proposed by Lixia Zhang and now is an IETF proposed standard (Braden et al. 1997). The RSVP provides two services: load-controlled service and guaranteed service. In Chapter II, we undertake a careful analysis of RSVP performance. Figure 2.3 shows its implementation architecture following (Braden et al. 1997).

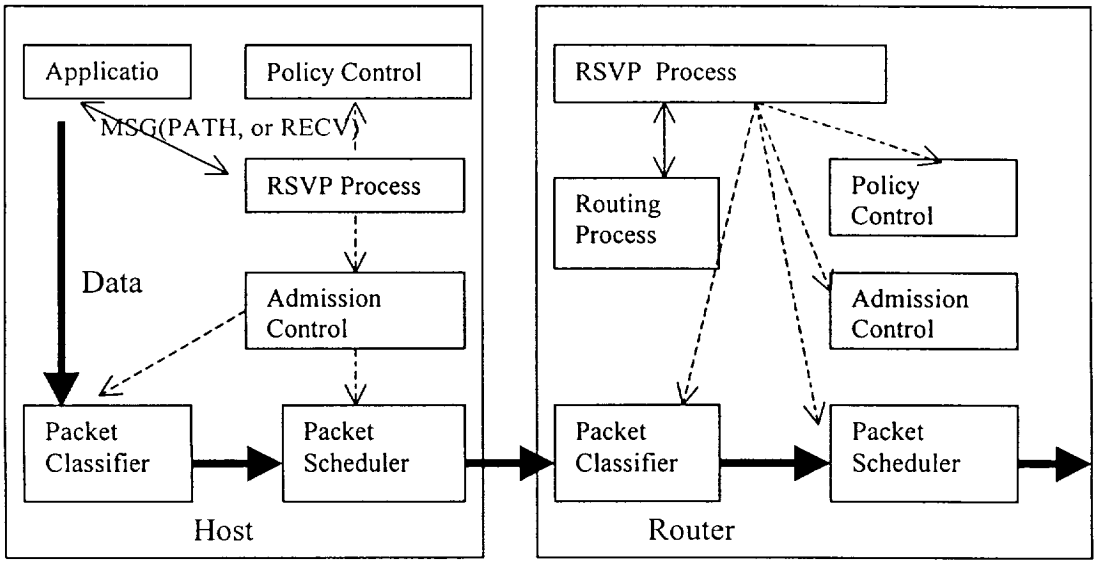


Figure 2.3. RSVP Integration in Host and Router

The RSVP process is implemented as a daemon process at the user-level. An application requests a certain quality of service from the RSVP daemon running on the host in terms of the sender side's traffic specification. The RSVP daemon then checks with an admission-control module to find out whether the node has sufficient resources to supply the requested QOS. If this node can afford the requested QOS, parameters are set in the packet classifier module and packet scheduler module to enforce the reservation. The RSVP daemon then sends the reservation request to the next node on the data path. This process continues to the destination node. If the admission test fails at any stage, the RSVP daemon sends an error notification back to the host. Once the reservation is accepted by every node on the data path, the RSVP flow is set up and will receive the requested quality of service. The packet classifier and packet scheduler modules on every node are jointly responsible for the quality of service given to a flow. The classifier looks at every data packet to determine whether the appropriate flow has a reservation and which QOS the flow should get.

The Tenet protocol suite was proposed by Ferrari, Verma and others. RSVP is only a signaling protocol and depends on other transport protocols to do data transfer while the Tenet protocol suite provides its own internet layer protocol called RTIP (Real-Time Internet Protocol) and two transport protocols called RMTP (Real-Time Message Transfer Protocol) and CMTP (Continuous Media Transfer Protocol). The Tenet protocol suite uses RCAP (Real-Time Channel Administration Protocol) as a resource reservation protocol. Figure 2.4 shows its software architecture (Banerjea et al. 1996). It coexists with the TCP/IP protocol and uses TCP for transferring control messages.

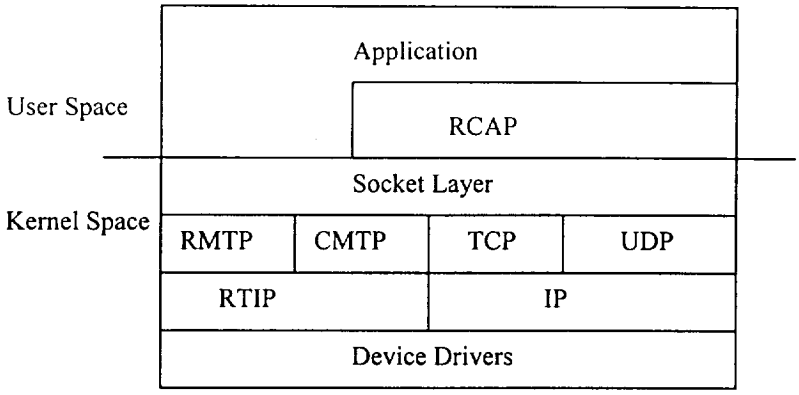


Figure 2.4. Software Architecture of the Tenet Protocol Suite

The Tenet protocol suite is intended for continuous-media applications. It provides deterministic, statistical service as well as best-effort service. Deterministic service is similar to guaranteed service while statistical service is similar to load-controlled service in RSVP. However, transport protocols RMTP and CMTP do not provide reliable data transfer. Even the deterministic service does not imply reliable service. If a packet is corrupted, the service model simply discards the packet since it is based on such an assumption that any mechanism to retransfer the packet will result in a missed deadline. Like RSVP, the Tenet protocol suite assumes that the link layer will provide bounded packet delay. Table 2.1 shows a comparison between RSVP and the Tenet protocol suite.

Table 2.1. Feature Comparison Between RSVP and the Tenet Protocol Suite

	RSVP	Tenet Protocol suite
Services Provided	Load-Controlled and Guaranteed services	Deterministic and Statistical services
Traffic Specification	Leaky Bucket model (r, b, p, m, M)	(Xmin, Xave, I, Smax)
QoS Specification	(r, b, p, m, M, g, s)	(Dmax, Zmin,Jmax, Wmin)
Transfer Protocol	TCP/UDP	RMTP/CMTP
Control Protocol	RSVP	RCAP
Reservation Initiator	Receiver	Sender
Reservation Sharing	Support	No support
Status	IETF Proposed Standard	Non-standard protocol

Communication Middleware

Middleware is considered to be an efficient software architecture for implementing a real-time distributed communication layer (Mizunuma, Shen and Takegaki 1996). It provides an application-programming interface and masks the differences between various underlying communication hardware. Communication middleware usually includes following features: a programming model, a real-time transport protocol on top of native services, a QoS mapping algorithm, user-level multiplexing schemes, and local and global admission mechanisms (Mizunuma, Shen and Takegaki 1996).

Middleware itself is configurable and selectable. Both the RSVP protocol and the Tenet protocol suite were implemented as middleware. On top of ATM, a real-time distributed computing middleware called MidArt was also implemented (Mizunuma, Shen and Takegaki 1996). Efficient middleware is usually tightly bound with the run-time environment so as to achieve high performance and predictability. The fundamental issue for middleware to provide sub-network-wide QoS is to have control over all endpoint systems and network elements within this sub-network.

Communication middleware with QoS mainly consists of a QoS mapper module, an admission-control module, and a packet scheduler module. The QoS mapper implements the mapping of user-level QoS and traffic parameters to network-specific QoS and traffic parameters. The admission-control module determines whether the currently available resources can accommodate a new request. It includes local and global admission control. Local admission control determines whether a local endpoint system has sufficient resources whereas global admission control checks whether all endpoint systems and switches in a sub-network have sufficient resources. Global admission control is realized through an admission-control protocol that defines a set of control messages to be passed between control entities along the end-to-end path. The packet scheduler is an implementation of service disciplines or queuing algorithms.

Summary

In this chapter, we reviewed the basic approaches for real-time communication in packet-switching network. Those approaches are intended for real-time communication in wide area networks, or the Internet. A common idea is to reserve resources, which

again is realized through packet scheduling algorithms. Required resources for a connection are calculated based on the user's QoS request and traffic descriptions of data sources. Admission-control mechanisms are employed to test whether the network can provide the requested QoS. The problem for such approaches is that, in a wide area network, it is hard to control network resources, which makes it difficult to achieve hard end-to-end QoS. A client-server-based model is evidently not suitable for real-time distributed computing either. Real-time distributed computing in a cluster requires hard end-to-end QoS and point-to-point, peer connections. Elsewhere in this thesis, we design and implement a real-time communication middleware to investigate how QoS can be efficiently provided and which factors affect QoS.

CHAPTER III

QUALITY OF SERVICE ON SUB-NETWORKS

In this chapter, we investigate quality of service on sub-networks. As discussed in Chapter II, the link-layer QoS support and sub-network-wide resource management on top of the network layer are two necessary requirements in order to achieve QoS in a closed sub-network.

Link-layer Protocols for Supporting Quality of Service

Real-time media access control (MAC) protocols for multi-access networks try to achieve real-time communication in multi-access networks. In a multi-access network, nodes communicate via a single shared physical link, and at any given time, only one node is allowed to access this physical link to send packets to another node or nodes. The dynamic reservation method is similar to Time Division Multiplexing (TDM). It has been adopted for use with both the Carrier Sense Multiple Access / Collision Detect (CSMA/CD) window protocol and a token-passing protocol (Malcolm and Zhao 1995). However, it requires a global clock in order to coordinate the access to the shared physical link, which makes the implementation of this method difficult to be exact because of the scheduling algorithms and priority arbitration protocols employed. A global clock has to be refreshed periodically. On the contrary, switching network technology allows all nodes to send and receive messages simultaneously at full link

speed. In a switching network, the packet-scheduling algorithm can be implemented inside the switch without global clock synchronization and their implementations can be exact. We investigate two link-layer protocols: ATM and IEEE 802.1p. Both protocols are intended to provide quality of service.

The IEEE 802.1p Protocol

The IEEE 802.1p protocol is a simple priority-based Media Access Control (MAC) protocol for switched Ethernet. It specifies both the setup of Virtual LAN (VLAN) information and the nature of traffic that will travel over the VLANs to support time-critical traffic for a switched LAN. The protocol achieves QoS through prioritization of traffic classes (IETF 1999). The IEEE 802.1p protocol also provides efficient support of multicasting. Usually packet delay inside a switch consists of queueing delay and access delay. Priorities in the IEEE 802.1p include queueing priority and access priority. It allows up to eight traffic classes, different priorities on different ports, and dynamic multicast filtering. The IEEE 802.1p protocol also supports priority designation to IEEE 802 MAC protocols. Combined with IEEE 802.1Q protocol (IEEE 1999), the IEEE 802.1p protocol facilitates QoS over Ethernet by providing a means for tagging packets with an indication of the priority or class of service desired for the packet. These tags allow applications to communicate the priority of packets to internetworking devices. RSVP support can be partly achieved by mapping RSVP sessions into the IEEE 802.1p service classes. One disadvantage of IEEE 802.1p protocol is that it allows only off-line priority designation.

Quality of Service in ATM

As opposed to switched Ethernet, Asynchronous Transfer Mode (ATM) is designed to provide end-to-end QoS on a per-connection basis through traffic management. An end-to-end connection can be established through cascading virtual channels. Traffic management enables an ATM network to deliver individual connections, as well as protect against conditions that could result in congestion and degraded performance. ATM works to achieve these goals by the following techniques (Zheng, Shin and Shen, 1994): 1) support for multiple types of traffic at different speeds; 2) satisfaction of each application's QoS requirements on a per-connection basis; 3) maximization of the utilization of network resources; 4) protection of ATM end-users and the network in order to achieve network performance objectives; 5) minimization of reliance on ATM Adapter Layer (AAL) and higher-layer traffic management schemes in order to reduce or eliminate congestion in an ATM network. In order to reach those goals, ATM will perform an admission test before it accepts a connection with a certain requested QoS. If a connection exceeds its negotiated traffic rate, the ATM network has the right to discard or tag those cells and notify end users. ATM networks also provide fair and equitable access for ATM end users wishing to use unused network resources on a best-effort basis.

The ATM network defines a service architecture consisting of five ATM service categories that relate traffic and QoS parameters to network behavior. These service categories are as follows: 1) constant bit rate (CBR); 2) variable bit rate, real time

(VBR-rt); 3) variable bit rate, non-real time (VBR-nrt); 4) available bit rate (ABR); and 5) unspecified bit rate (UBR). Real-time communication on ATM must be mapped to

Table 3.1. Traffic and QoS Parameters in ATM Service Categories

Attributes for Traffic parameters and QoS parameters	ATM Layer Service Categories				
	CBR	VBR-rt	VBR-nrt	UBR	ABR
Traffic Parameters					
Peak Cell Rate (PCR) Cell Delay Variance Tolerance (CDVT)	yes	yes	yes	yes	yes
Sustainable Cell Rate(SCR), Maximum Burst size(MBS) CDVT	N/A	yes	yes	N/A	N/A
Minimum Cell Rate(MCR)	N/A	N/A	N/A	N/A	yes
QoS parameters					
Max End-to-End Cell Delay Variance(CDV)	yes	yes	no	no	no
Maximum Cell Transfer Delay (CTD)	yes	yes	no	no	no
Cell Loss Rate (CLR)	yes	yes	yes	no	no

service categories of CBR or VBR-rt. Each service category has its own QoS and traffic specification. Table 3.1 shows these ATM services following (Shen 1996).

Each service category corresponds to an ATM adaptation layer. AALs sit on top of the ATM layer. Their main purpose is to adapt the flow of information received from a higher-layer application like voice or data to the ATM layer. Each AAL consists of two

sub-layers: the segmentation and reassembly sub-layer and the common convergence sub-layer. Except for UBR, the other four service categories are based on connection-oriented mode of operation. AAL 1 and AAL 2 provide CBR and VBR-rt services, respectively. AAL 3, 4 and AAL 5 support services of VBR-nrt and UBR. AAL0 supports ABR (McDysan and Spohn 1995).

Though ATM network architecture can provide the end-to-end QoS guarantee for CBR and VBR-rt, ATM is still in its infancy because it lacks standard distributed Cell-Admission-Control (CAC) algorithms, and efficient cell scheduling algorithms. Middleware that bridges applications and ATM services is needed (Shen 1996). One of IETF efforts is on how to map QoS defined in RSVP on to ATM (Crawley, Berger, Berson, Baker, Borden and Krawczyk 1998).

Experiments on IEEE 802.1p

Experimental Testbed

In this experiment, we use one Extreme brand gigabit switch to connect five hosts, each host in our testbed is a Sun Ultra-SPARC workstation running Solaris 2.6 operating system. The testbed is illustrated in Figure 3.1.

The link between host and switch supports full duplex 100 megabit per second bandwidth. Multiple switches can be connected together through its gigabit-per-second Ethernet port. So, essentially, arbitrary topologies are possible. The Extreme brand gigabit switch implements the IEEE 802.1p protocol. However it does not support dynamic and connection-based priorities. Four priority classes are supported in Extreme

gigabit switches: low, normal, medium, and high. Inside a switch, each port is actually associated with a queue. The switch uses this priority to schedule packets. When traffic conflict occurs, traffic from the high-priority queue gets passed first. Obviously, such a simple strategy will completely starve low-priority traffic if the traffic from the high-priority queue lasts for a sufficiently long time.

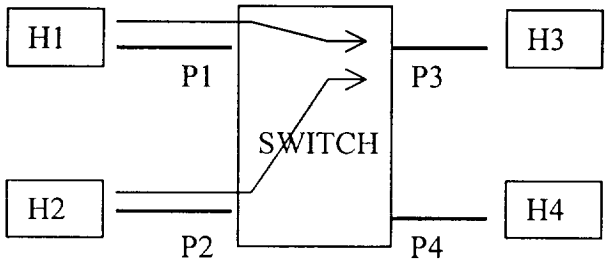


Figure 3.1. Sub-Network Configuration for IEEE 802.1p Experiment

Purpose and Procedures

The purpose of this experiment is to test whether the link-layer protocol IEEE 802.1p provides priority-based QoS guarantees and to measure how well this protocol works for time-sensitive traffic class, or to which degree QoS can be achieved using this protocol.

In this experiment, nodes H1 and H2 send traffic to the third node H3 at wire speed (see Figure 3.1). Since each link speed is 100 megabit per second, eventually, traffic conflict will occur at port 3. We undertook two tests. In test 1, we assigned the same priority to port 1 and port 2. In test 2, we assigned high priority to port 1 and low

priority to port 2. Table 3.2 lists what each experiment tests and their respective descriptions.

The results of test 1 are presented in Figure 3.2 and Figure 3.3. Results of test 2 are presented in Figure 3.4 and Figure 3.5. Figure 3.2 shows the end-to-end delay for TCP connection 1 (from H1 to H2) and Figure 3.3 shows the end-to-end delay for TCP connection 2 (from H2 to H3). The switch monitor showed that there was no packet corruption during the life time of these two connections.

Table 3.2. Experimental Descriptions for IEEE 802.1p

Tests	Purpose	Experimental Descriptions
Test-1	To test whether the IEEE 802.1p protocol provides fair service to connections with same priority	Create multiple connections from different sources to the same destination. Specify all connections same priority. Each source sends packets at wire speed. So conflict will eventually occur at the output port of the switch connected to the destination node. Packet delay on each connection is recorded. Figure 3.2 and Figure 3.3 show the case in which two connections are set up.
Test-2	To test whether the IEEE 802.1p protocol provides expected priority service.	Similar to Test1, but assign each connection with different priorities. The output port of the switch is supposed to serve each connection in terms of its source port's priority. Packet delay will be recorded. Figure 3.4 and Figure 3.5 show the case in which two connections are set up.

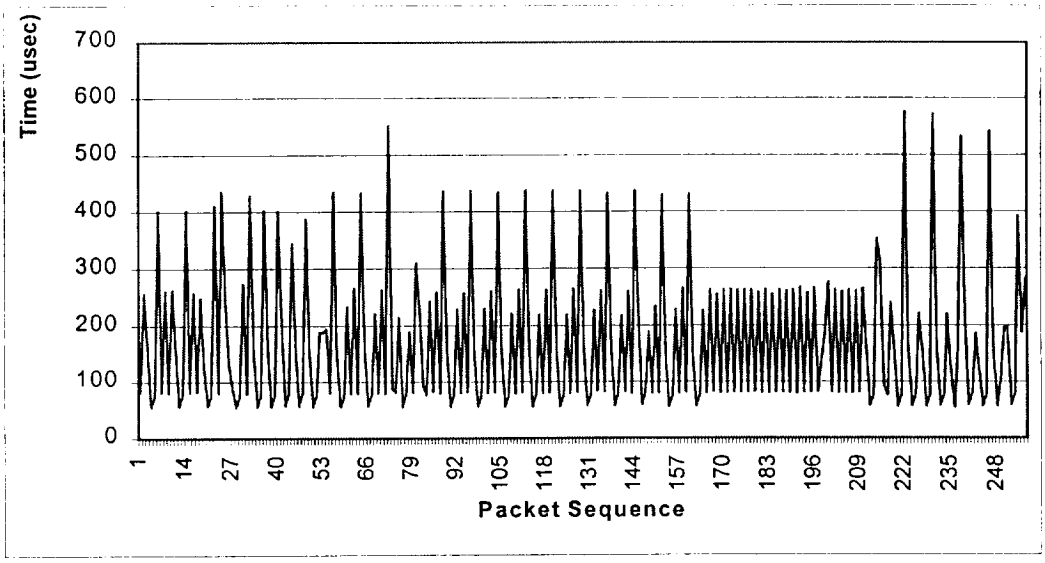


Figure 3.2. End-to-End Packet Delay on Connection 1, Test 1

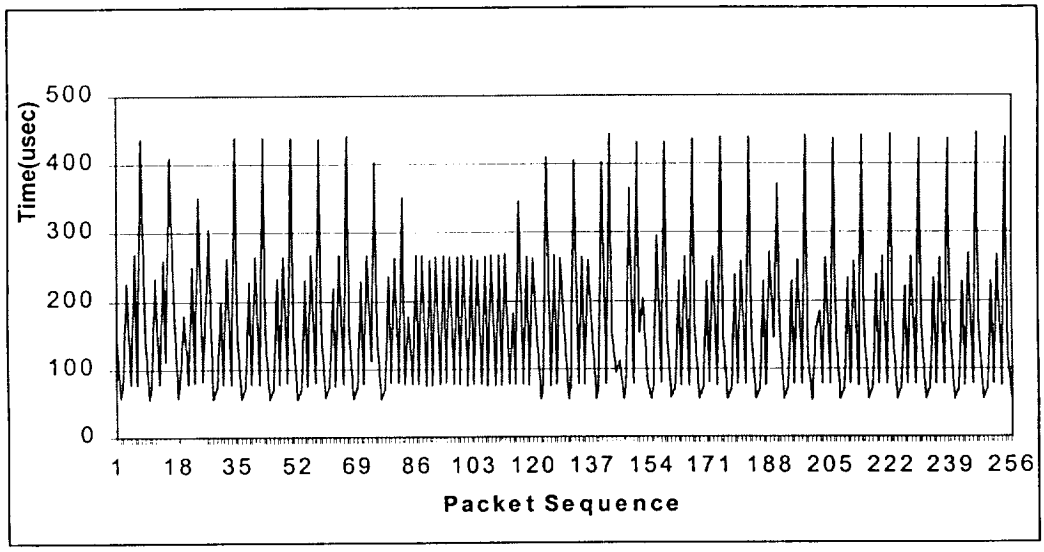


Figure 3.3. End-to-End Packet Delay on Connection 2, Test 1

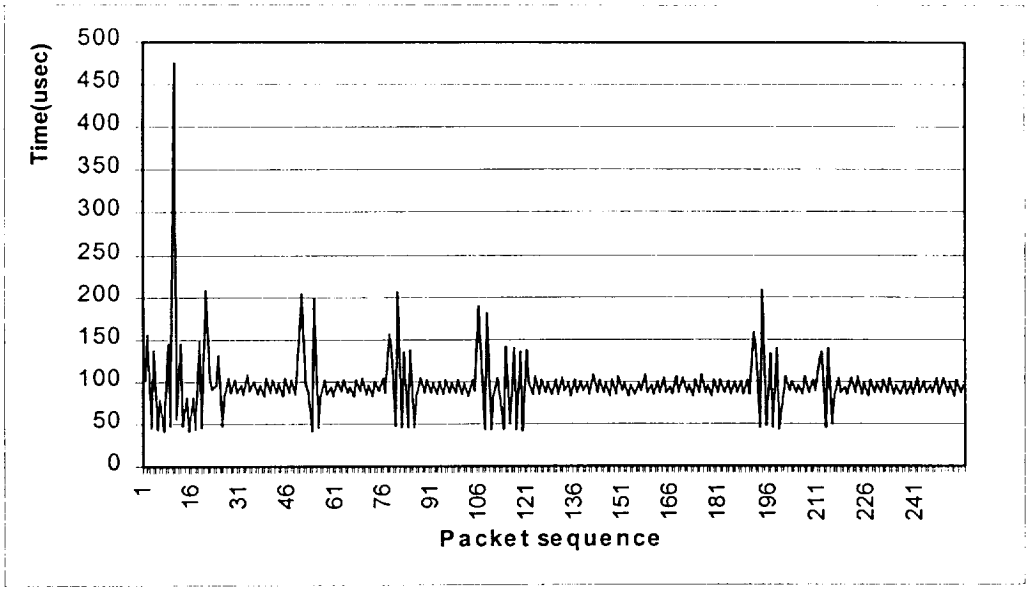


Figure 3.4. End-to-End Packet Delay on Connection 1, Test 2

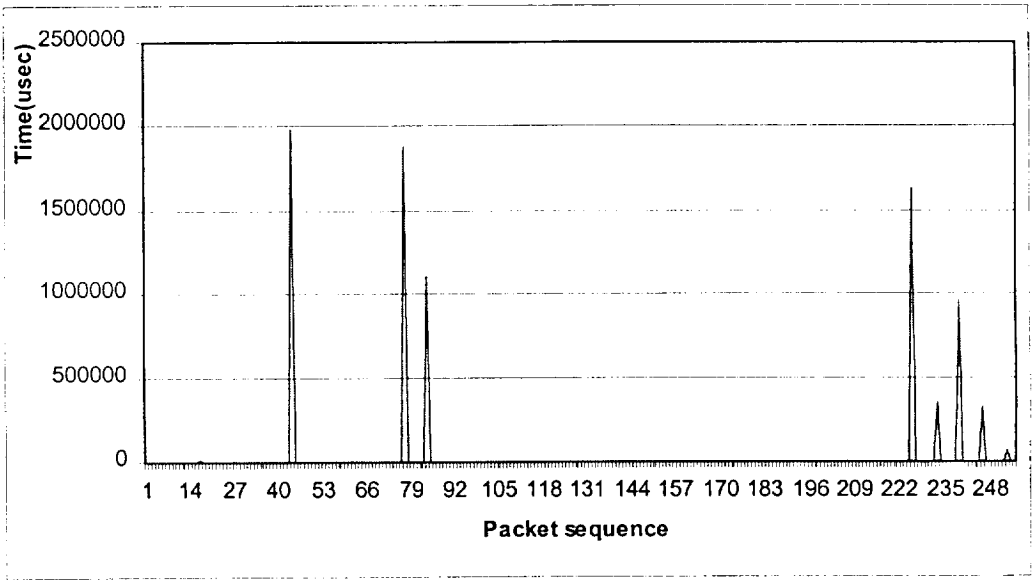


Figure 3.5. End-to-End Packet Delay for Connection 2, Test 2

Analysis of Results and Significance

From test 1, we find that the average end-to-end delays for a 1K packet on connection 1 and connection 2 are 171.25 (μsec) and 170.38 (μsec), respectively. Their respective bandwidths are 46 megabit per secondand 47 megabit per second. Since the physical link's raw bandwidth is close to 100 megabit per second, the switch provides two connections that are almost perfectly sharing the bandwidth. Different packet sizes affect only end-to-end packet delay; bandwidth is still perfectly shared. The maximum end-to-end delay variance is nearly 600 μsec. The delay variance is mainly caused by protocol processing and context switching inside the operating system. From test 2, we can see that connection 1, which is from the high-priority port 1, gets almost the entire bandwidth of the link. Connection 2, which is from the low-priority port 2, get serviced only when there is no traffic from the high-priority connection, which results in huge delay fluctuations on connection 2. This experimental result means that the switch blocks the packets from the low-priority port when conflict transpires.

From these two simple experiments, we can conclude that the IEEE 8021.p protocol provides reasonable bandwidth sharing among connections with the same priority. However, end-to-end packet transfer delay is not guaranteed for a packet from the low-priority port. Because the operating system cannot give the packet receiver and sender guaranteed CPU time, extra delay jitter can be observed during the life of each connection. Even for the high-priority connection, its end-to-end delay is not smooth (see Figure 3.5), which proves part of our research hypothesis. That is, guaranteed CPU

processing power at endpoint systems for handling data traffic is indispensable to achieving hard end-to-end quality of service. Or, in other words, local admission control at the hosts (endpoints) becomes necessary. The low-priority connection is starved because traffic conflict inside the switch occurs and the switch provides only simple, priority-based service. There are two approaches to avoid traffic conflict. One approach is to use a global admission-control mechanism in order to guarantee that no traffic conflict will occur. The other method is that the switch itself provides an admission mechanism or participates with endpoints in the admission process.

Resource Reservation Protocol

The Resource ReSerVation Protocol (RSVP) was jointly proposed by the Information Sciences Institute of the University of Southern California (USC ISI) and Xerox Corporation's Palo Alto Research Center (PARC). Now RSVP is a proposed standard of the Internet Engineering Task Force (IETF) (Braden et al. 1997).

RSVP is intended to provide QoS in wide-area networks or the Internet. The targeted applications are video and audio applications that last a long time. It is not suitable for short-lived connection applications such as ftp, web access, telnet, and so on, since the overhead of setting up an RSVP flow cannot be fully justified for such scenarios. RSVP is proposed as a supplement to the current TCP/IP-based network model. A TCP/IP-based network provides only best-effort service in which the network promises not to delay or discard packets intentionally and does its best to forward packets to the next hop or destination. RSVP itself is just a signaling protocol. It sets up a reservation at each node along the path, but enforcement of the reservation must be

done by the packet scheduler and classifier at each node. It is the enforcement of the reservation that brings QoS to user applications. It is a fallacy that RSVP itself will provide QoS. RSVP can reserve resources on a unicast connection or multicast tree. It depends on other transport protocols (typically TCP or UDP) to transfer the actual data.

An RSVP flow can request load-controlled service or guaranteed service. A flow is an end-to-end connection and is equivalent to the channel concept in the Tenet scheme. Load-controlled service provides a statistical guarantee and is essentially priority-based service. The end-to-end performance depends on the total traffic inside this traffic class and the available bandwidth. Guaranteed service attempts to provide hard end-to-end QoS. RSVP contains a policy control mechanism that determines which entities can make a reservation. Authentication, access control and accounting are ongoing research topics (Braden et al. 1997).

Experiments on RSVP

Experimental Purpose and Procedures

The purpose of measuring RSVP is to show how well load-controlled service provided by RSVP performs in a closed sub-network, and also to show that RSVP itself cannot provide an end-to-end delay bound if the underlying link-layer protocol does not provide bounded delay. The experimental results can partially prove our hypothesis and conforms to our theoretical analysis. We perform our tests on an RSVP implementation that supports load-controlled service.

The experiment is based on same testbed used in testing the IEEE 802.1p protocol. The RSVP middleware is installed on each host of the sub-network. In the first test, we create a single RSVP flow with specified QoS using TCP as a transport-layer protocol, and measure its end-to-end packet delay. Then we compare its end-to-end delay with a simple TCP connection to show whether the RSVP introduces extra overhead by its soft state refreshment mechanism and packet scheduler and packet classifier. In the second test, we create multiple competitive RSVP flows, and observe whether RSVP can provide load-controlled service.

Analysis of Results and Significance

Figure 3.6 represents the end-to-end packet receiving delay of an RSVP flow using its load-controlled service. The receiver requests 100 megabit per second average rate and peak rate is same as average rate. The bucket size is also 100 megabit per second. Packet size is 1K bytes. Within a long period, we sample 256 consecutive packets. We find that the average receiving delay per packet is close to the ideal value of 100 μsec. The receiving delay for the most of packets is approximately 100 μsec. As for receiving delay jitter, we think that there are two causes: 1) kernel buffering for TCP/IP protocol processing and 2) context switching. The first is due to the fact that we use RSVP flow. The TCP connection does not differentiate the data boundary.

When we compare Figure 3.6 with Figure 3.5, in which a simple TCP connection with full bandwidth is created and is assigned with highest priority, we see that the

causes that result in delay jitter are the same. The difference is that RSVP provides an RSVP flow user-specified rate. The Figure 3.6 perfectly reflects this point.

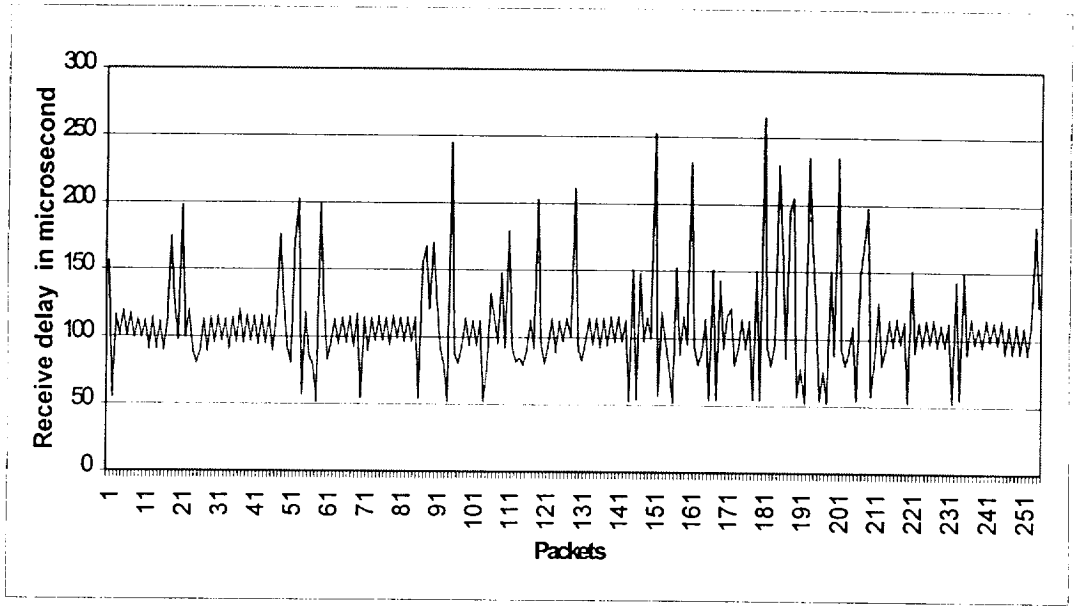


Figure 3.6. Packet Receiving Delay in Single RSVP flow

Figure 3.7 represents end-to-end receiving delay of an RSVP flow with both average rate and peak rate equal to 72 megabit per second. However at the sending side, there is another RSVP flow with sending rate equal to 18 megabit per second. We can see that RSVP provides fair sharing of bandwidth because the average receiving delay for the high-speed RSVP flow is 110 μsec. This means that RSVP provides load-controlled service but results in more frequent packet delay jitter. This experimental result means that more frequent context switching occurred at the send-side host.

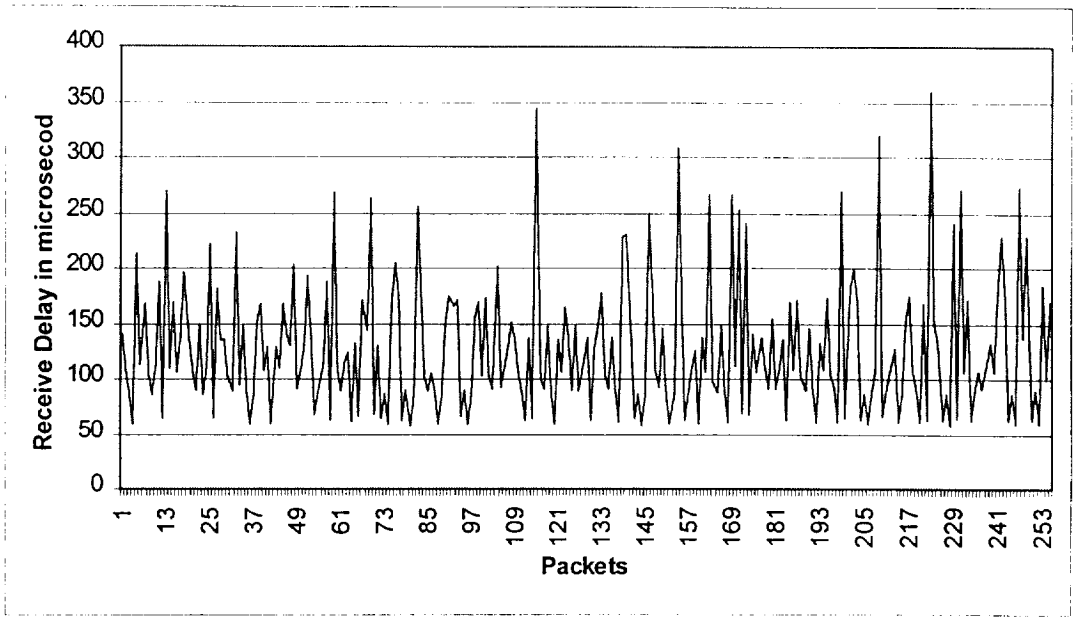


Figure 3.7. Packet Receiving Delay in Multiple RSVP flows

For the case in which multiple RSVP flows are created, packet-access conflict could happen inside switches if a switch does not have RSVP enforcement. The switch will not reject a new RSVP flow even when the sum of bandwidth requirements of all RSVP flows is beyond its capacity. So from the experiments on RSVP, we have proven our research hypothesis that not only must the endpoint system provide guaranteed CPU time for handling packet transfer, but also sub-network-wide resource reservation is indispensable to provide conflict-free access inside switches and so achieve smooth end-to-end delay.

We also found that RSVP flow introduced extra overhead on endpoint systems because a reservation must be refreshed periodically to avoid the flow to be torn down (Braden et al. 1997). Resources in RSVP are automatically released if the reservation

request is not refreshed. However, for a sub-network environment, keeping the flow state becomes unnecessary because the routes will never change during the life of the flow. In addition, some features such as policy control are unnecessary for sub-network application because users have complete control over the sub-network. In the next chapter, we present our communication layer design with a more efficient resource reservation mechanism.

From the beginning, RSVP was designed to run on the IP protocol and as a signaling protocol for resource reservation. This implies that it cannot be an optimized method for a particular sub-network. An ongoing effort is RSVP on ATM. The purpose is to integrate RSVP signaling and ATM signaling in support of Integrated Services (Crawley et al. 1998). It involves two issues: QoS mapping from RSVP QoS model to ATM QoS model and virtual channel (VC) management. Obviously if RSVP can directly use the connection-oriented QoS of ATM network, guaranteed service can be efficiently provided in high-speed sub-networks based on ATM. Even when an RSVP implementation provides guaranteed services, it still has to depend on the underlying link layer in order to provide bound delay; otherwise hard end-to-end QoS cannot be provided.

CHAPTER IV

THE DESIGN AND IMPLEMENTATION OF COMMUNICATION MIDDLEWARE WITH QUALITY OF SERVICE GUARANTEES

In this chapter, we describe the design and implementation of our communication layer. Whereas RSVP is inefficient in managing resources for QoS communication in high-speed sub-networks, our communication layer design and implementation provide a more efficient resource management middleware.

Application Programming Interface Design

After carefully investigating the application programming interfaces (API) provided by RSVP and the Tenet scheme, we decided that a scheme like RSVP using explicit client-based reservation is not a good choice, because in a real-time distributed computing environment each process can be a server and also as a client. It consequently requires peer-to-peer communication (Arvid 1991). In RSVP and the Tenet scheme, an implicit assumption is that a channel source or destination is a client, and only the server handles multiple clients. It also implies that a client needs only limited CPU processing power compared to the server. But in a high-speed sub-network environment, all endpoint systems usually have the same or comparable processing power.

Table 4.1 lists the main API for our communication layer design. From Table 4.1, we can see that the basic functions are simple and bear some characteristics of MPI/RT (MPI/RT Forum 1998) and MPI (MPI Forum 1994).

Table 4.1. Communication Interface Functions

int RT_Channel_create(int Src_rank, int Dst_rank, QOS_t *qos, CHANNEL_t *channel)
int RT_Channel_delete(CHANNEL_t channel)
int RT_Channel_modify(QOS_t *qos, CHANNEL_t *channel)
int RT_Channel_status(CHANNEL_t channel, CHANNEL_STATUS_t *status)
int RT_Putmsg(CHANNEL_t channel, char *msg, int size)
int RT_Getmsg(CHANNEL_t channel, char *msg, int *size)
int RT_Init(int argc, char *argv[])
int RT_Finalize()
int RT_Get_rank(int *rank)

We view a channel's traffic is a part of QoS so as to simplify the definition. Only one data type (character string) is supported because it is sufficient for investigating QoS in sub-networks. Once a point-to-point channel is created, sending and receiving messages will be under control of the QoS. Any channel that violates QoS will result in messages being lost. *Channel_t* is an opaque object and is implementation dependent. The function *RT_Init* creates a communication context, activates the packet scheduler, and determines the available resources in an endpoint system and sub-network-wide resources in terms of current configuration. Numbering each process's rank is done inside this function; each process has a unique rank that is generated in terms of a configuration file. In our implementation, we use a control thread to manage the creation, deletion and modification of channels. The packet scheduler is also a bound

thread. *RT_Finalize* will free all dynamically created system resources. After the *RT_Finalize* function call is invoked, receiving and sending messages are not allowed. However *RT_Finalize* will wait for all pending messages to be finished. The RSVP API consists of four functions: *rapi_reserve*, *rapi_sender*, *rapi_session* and *rapi_release* (Braden et al. 1997). These functions are similar to the Berkeley socket interface. Our channel creation function combines RSVP's *rapi_reserve* and *rapi_sender* since our model is based on peer-to-peer communication.

Implementation Description

Our implementation was accomplished on the Sun Solaris 2.6 operating system. Figure 4.1 illustrates the implementation framework.

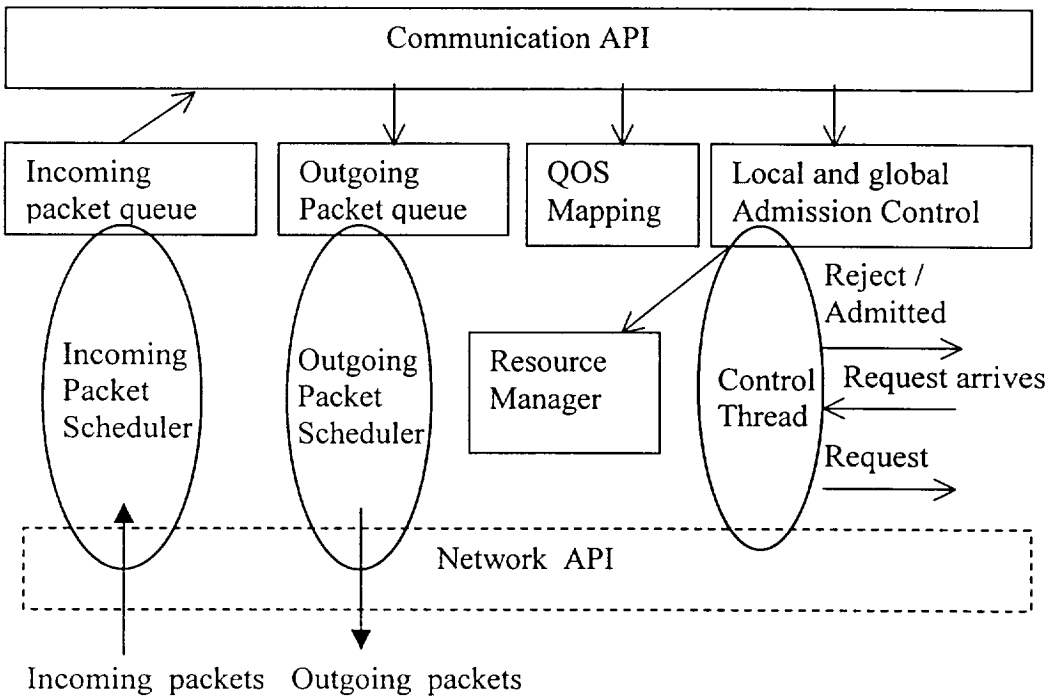


Figure 4.1. Communication Middleware Architecture

A control thread behaves like a daemon process. It receives control messages from other processes or sends out control messages to other processes. Control messages include creation, modification, deletion and status request of a channel, and so on. Each process has a control thread. The thread running on rank 0 process is a master-control thread, which is responsible for the allocation and management of global shared resources. Control messages from non-master control threads (slave control threads) are first sent to the master control thread, which will forward the control messages to other control threads if the control messages are not intended to this process. In this way, the master thread will have knowledge of all created channels so that it can do global admission control.

Our implementation of the Class Based Queuing (CBQ) algorithm is straightforward. Each channel will be classified in terms of its priority. The CBQ scheduler will serve each priority class in round robin. In each class, channel will be served in FIFO.

Channel establishment involves two phases. In the first phase, channel establishment does its local admission test, and sends out the channel creation message to the remote endpoint of the channel through master control thread. The second phase is to wait for confirmation from the remote endpoint. Only when both endpoints pass their local admission tests will a global admission test be conducted. If the global admission test is also passed, this channel is established. Figure 4.2 shows the procedure.

Completely different from RSVP and the Tenet admission tests, our method needs only three main tests; it is not a hop-by-hop-based method since we limit our middleware to a closed sub-network. In addition, our scheme requires that both endpoints initiate channel reservation since our middleware is based on peer-to-peer model.

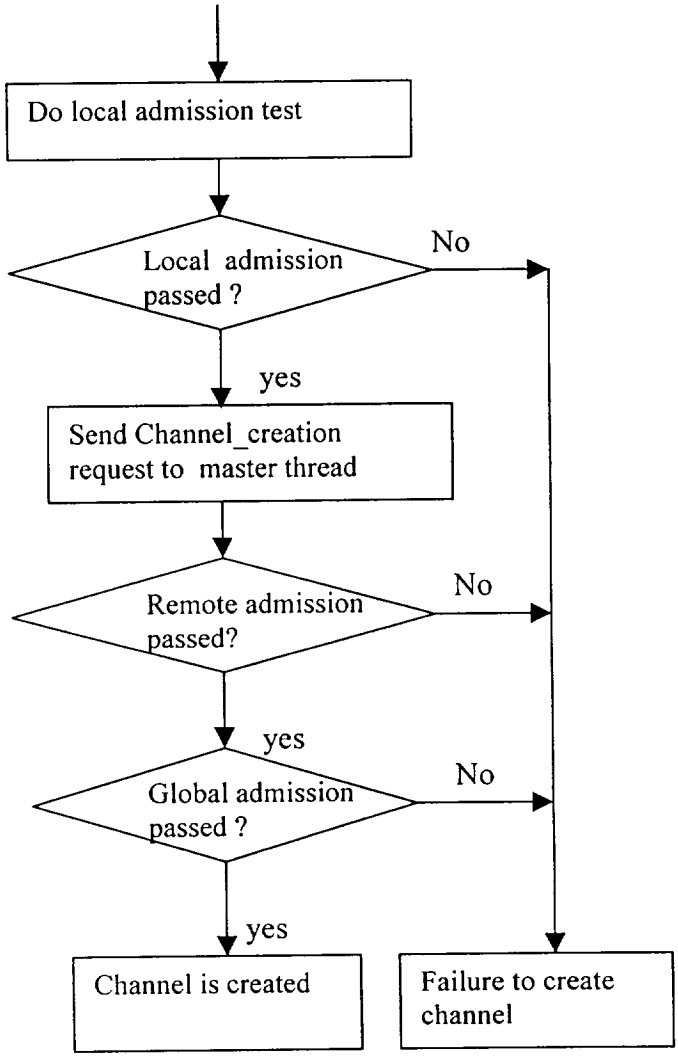


Figure 4.2. Flow Chart For Channel Creation

The channel scheduler is a bound thread for sending packets to the network. It implements the CBQ algorithm and a queue is associated with each channel. The queue size is determined by the QoS parameters during admission test. All queues will be served by the scheduler during a quota of time that is ascertained during admission. In our current implementation, priority and bandwidth percentage are supported and the time quota is calculated in terms of the bandwidth percentage requested.

A more efficient way for channel admission is to use a collective admission-control mechanism through a commit operation. Once committed, all requested channels are created. The MPI/RT standard uses this mechanism (MPI/RT Forum 1998). A collective admission-control mechanism also improves resource usage and the probability that a channel can be admitted. The process of modifying a channel is similar to the channel creation. Reducing quality of service will guarantee the success of modification of a channel. Deletion of a channel will wait for the messages in the queue associated with the channel at the sending side to be sent out before this channel is removed. Deletion of a channel also needs a two-phase procedure.

When the application calls ***RT_Init***, the resource object at each process will be initialized in terms of user-provided configuration parameters. At the master process, it also contains the descriptions of global resources such as network topology, switching bandwidth, and so on. Besides creating a global communication context, its tasks also include creating the daemon scheduling thread and the daemon control thread.

Figure 4.3 represents QoS structure used in the current design. Currently time-driven QoS is not implemented since we do not have a hard real-time operating system.

The QoS definition also includes traffic definitions like period and minimum inter-arrival time for non-period message stream. Traffic parameters are viewed as QoS parameters. The QoS definition in this design is based on reliable data transfer. If a statistical guarantee is needed, a probability parameter for timely delivery should be added to the QoS parameter (a QoS formulation in which those packets that miss their deadlines will be discarded). Loss-rate can also be added as one of the QoS parameters since continuous media applications can tolerate loss-rate to a certain degree. For the purpose of measurement, we associate a status structure with the channel so that the application can get feedback about the current channel. The status query is important for getting the desired QoS from the system.

At the beginning, the user usually has no idea how much QoS the system can provide. The Status parameters include those actual QoS values achieved by the channel, such as minimal inter-arrival time till the present time and average packet-arrival time. It is also useful for the adaptive admission-control algorithm. Thus, performance feedback from experience can be used to achieve higher utilization. In our current implementation, the QoS mapper is straightforward since we did not directly use QoS that the link-layer protocol provides.

```
typedef struct _QOS_PRI
{
 int    pri;              /*  channel's priority              */
 int    bandwidth;        /*  required bandwidth percentage   */
}QOS_PRI;

typedef struct _QOS_TIME
{
 float  dmax;      /* End-to-End delay upper bound             */
 float  jmax;      /* End-to-End jitter upper bound            */
 int    smax;      /* the Maximum message size                 */
 float  period;    /* the message arrival period               */
 float  xmin;      /* the minimum message arrival interval for
                      non-period message transfer              */
}QOS_TIME;

typedef struct _QOS
{
 int    QosType;   /*  Two types of QOS                        */
 union
  {
    QOS_TIME  _qosTime;
    QOS_PRI   _qosPri;
  }qos;
}QOS_t;
```

Figure 4.3. QOS Structure

CHAPTER V

MEASUREMENT AND ANALYSIS OF QUALITY OF SERVICE

This chapter presents the experimental designs and measurements of the communication middleware that was described in Chapter IV. We offer detailed performance analyses according to the experimental results. We measure the end-to-end delay, delay jitter, and various effects caused by middleware, operating system and networks to prove the research hypothesis.

Design of Experiments

In Chapter II, we said that bandwidth, delay, and jitter are the three most important metrics of QoS communication. Bandwidth can be derived from packet size and packet transfer delay. Loss rate is not concerned in our measurement since our implementation is based on reliable communication. In addition to QoS, the sub-network-wide admission-control strategy and the CBQ algorithm are evaluated.

One difficulty in analyzing the performance is that our communication layer runs on a non-real-time operating system. POSIX thread implementation on Solaris operating system does not support priority scheduling. There is no way to get guaranteed CPU execution time for a bound thread. However we can find out the basic thread-scheduling period from the operating system. Through a tracing of the thread-

scheduling context, we can determine whether a light-weight process (LWP) context switching occurs. Table 5.1 lists the experiments.

Table 5.1. Communication Layer Measurement Experiments

Experi -ments	Purpose	Description
Exp-1	Measure end-to-end packet delay and jitter.	Create multiple channels under various conditions (packet size, priority, and bandwidth requirements) among a set of nodes. A typical case is that one process create two or more channels with other nodes.
Exp-2	Measure bandwidth reservation and sharing.	Create multiple channels among a set of nodes and measure each channel's obtained bandwidth. In the case of multiple channels sharing a link, each channel should get its expected bandwidth.
Exp-3	Test sub-network-wide admission control	Create various network load to test whether admission control correctly reject or admit a new requested channel. Typical case is that when the accumulated bandwidth of a set of channels is beyond the capacity of their shared physical link, or the switch capacity, then admission test should fail.

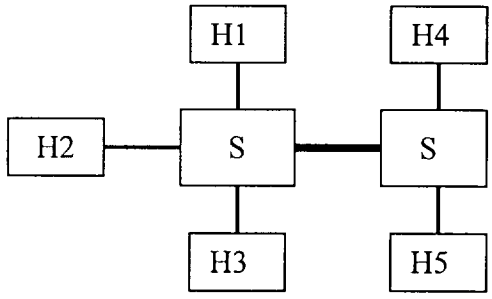
The first experiment is a delay and jitter test. In this experiment, the test program will measure sending and receiving delay under various conditions. Those conditions include varying packet sizes, priorities, and bandwidth requirements. A typical case is that one node has two or more real-time channels with other nodes. Each channel has its own QoS requirements. By measuring the each channel's end-to-end delay and delay jitter, we can find out whether middleware can provide the expected QoS. If not, we will track which factors cause abnormal delay and delay jitter. Another typical case is that a node has multiple incoming channels from different sources.

The second experiment is a bandwidth reservation and sharing test. In our implementation, the CBQ algorithm is used. Through creating multiple channels among multiple nodes and measuring each channel's bandwidth, we can experimentally show whether the CBQ algorithm provides expected bandwidth sharing and constraint delay. QoS implementation in our middleware will let users specify bandwidth percentage and priority.

The third experiment is a sub-network-wide admission test. Sub-network-wide bandwidth reservation guarantees that there is no traffic conflict inside switches, which is critical for middleware to provide hard end-to-end delay constraints. Actually, all measurement programs written for the first and second experiments involve sub-network-wide admission testing. Measurement programs in this experiment create channels that will construct traffic conflict inside switches and see whether admission control will detect this case and reject new admissions. Traffic conflict can also be monitored directly from switches.

We expect that our middleware implementation would provide better end-to-end delay and delay jitter than RSVP because of the simplified protocol processing and lack of soft-state refreshment mechanism. The CBQ algorithm implementation can provide expected bandwidth sharing. Strict end-to-end delay cannot be obtained because of context switching associated with thread scheduling, but within a scheduling period, delay variance should be bounded. Sub-network-wide bandwidth reservation ensures that no traffic conflict occurs inside switches.

The experimental testbed is illustrated in the Figure 5.1. It is similar to the one described in Chapter III. Here we use two Extreme brand gigabit switches. The link between host and switch supports full duplex 100 megabit per second bandwidth. The link between two switches supports gigabit per second bandwidth. Five Sun Ultra-SPARC workstations are connected to these two switches. The workstations run the Sun Solaris 2.6 operating system.



S: Extreme gigabit Ethernet switch
H: Sun Ultra-SPARC workstation.
——— 100 megabit per second link
▬▬▬ 1 gigabit per second link

Figure 5.1. Experimental Testbed

Analysis of Experiment Results

Figure 5.2 shows receiving packet delay jitter at a channel endpoint. In this experiment, one process has two outgoing channels connected to other two processes. All processes run on different nodes within a sub-network. We first let traffic generators produce slow traffic so that the scheduler will have sufficient time to process the arriving packets with an appropriate queue size. Admission test will guarantee that once a channel is admitted, the queue will never overflow if the sending side keeps its

promised traffic rate. Two channel sources produce a volume of 800K bits per second of traffic. Each channel requests 50% of bandwidth and each has the same priority. Figure 5.2 shows receiving delay jitter of 100 consecutive packets that were randomly sampled. During the whole life of the two channels, no queue overflow was observed. This means that the packet scheduler is fast enough to handle all incoming packets. In our implementation, once queues overflow, the input streams will be blocked.

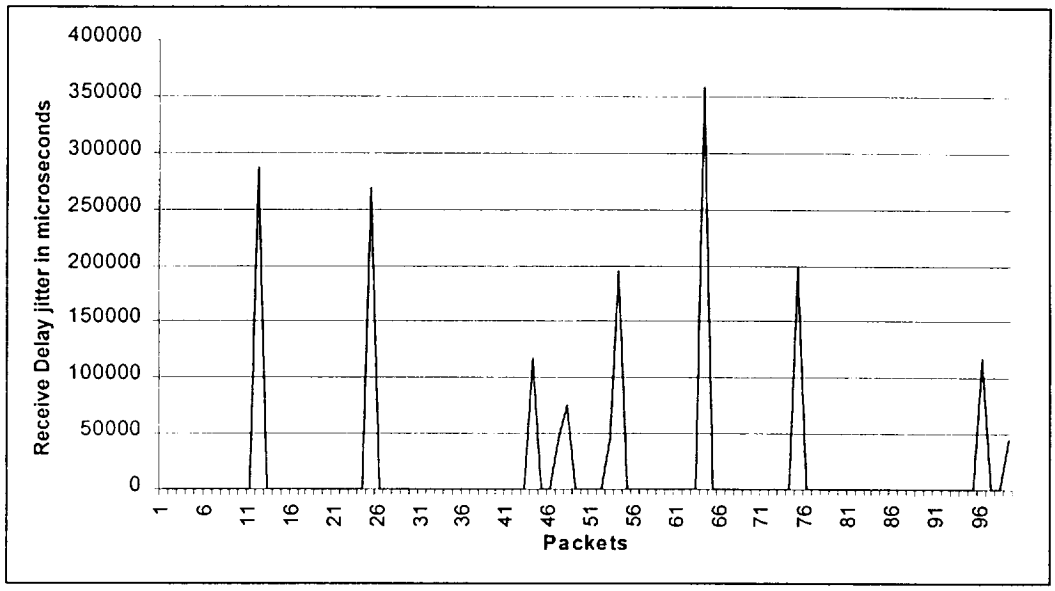


Figure 5.2. Receiving Delay Jitter with Interruption From Packet Generator, Test 1

From Figure 5.2, we see that receiving 12, 25, 44, 49, 64, 75, 96, and 99 packets cause huge delay jitter when compared to other points. Through tracing the thread scheduling point, we found that those points are exactly located at thread context-switching points. The operating system switches out the packet scheduler and runs the packet generator. However, in many practical systems, especially embedded systems,

the input data source is usually done by an external device and does not compete for CPU resources. So we may ignore the delay caused by the packet generator. Figure 5.3 is the case in which all thread-switching points are removed.

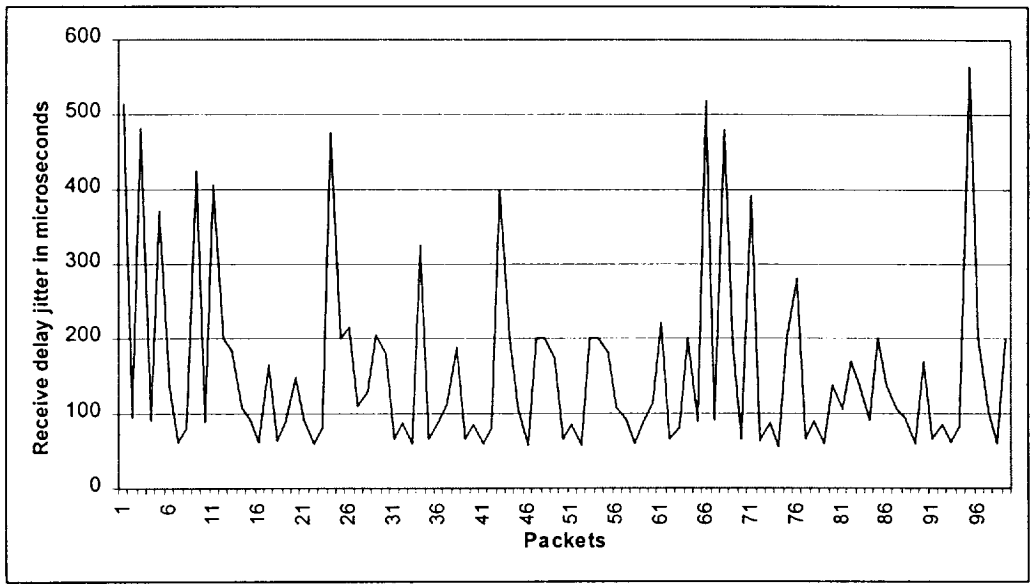


Figure 5.3. Receiving Delay Jitter Without Interruptions, Test 1

We did not find a smooth delay jitter either. Through further tracing of the behavior of the scheduler, we found that the jitter in Figure 5.3 was caused by the scheduling algorithm itself. In order to maintain bandwidth sharing, the implementation of CBQ uses the system function *gettimeofday* to obtain the current time and to compute the elapsed time and then calculate actual bandwidth that this channel obtains. The scheduler alternatively serves each channel. We observed that when the CBQ packet scheduler serves multiple packets within a scheduling period, delay jitter among those packets are smooth. The other factor that causes receiving delay jitter is that, at the

receiving side, the ***RT_get_msg*** call may be interrupted by thread context switching. From two channel's average delay jitter (157 μsec and 150 μsec, respectively), the CBQ packet scheduler does provide fair sharing of bandwidth.

Then we increased the packet generation rate to 1,600K bits per second and re-ran the measurement program. The effect caused by the thread scheduler was still observed. However, we found that only three receiving packets were affected by thread context-switching within 100 consecutive packets (see Figure 5.4 and Figure 5.5). This means that there are more packets available within a scheduling time slot for a channel because of the fast packet generation rate. If no packet is available within a packet-scheduling period, the scheduler will be in an idle state. The small jitter noted is mainly caused by thread context switching at the sending side or receiving side since the middleware can experience thread context switching between the scheduler thread and other system programs. The above experiments can partially prove our hypothesis that smooth end-to-end delay could be obtained if input packet queue never overflows and the scheduler gets guaranteed CPU processing power and is not interrupted by other processes or threads. We also measured delay and delay jitter in the situation that multiple channels are active simultaneously; the observed results are same as the two-channel case.

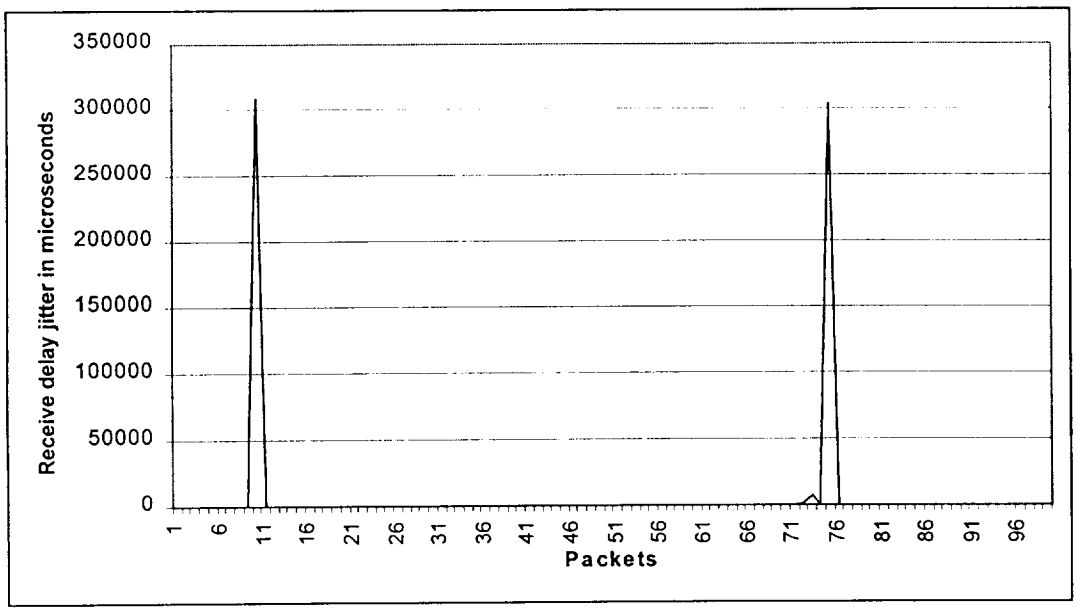


Figure 5.4. Receiving Delay Jitter With Interruption From Packet Generator, Test 2

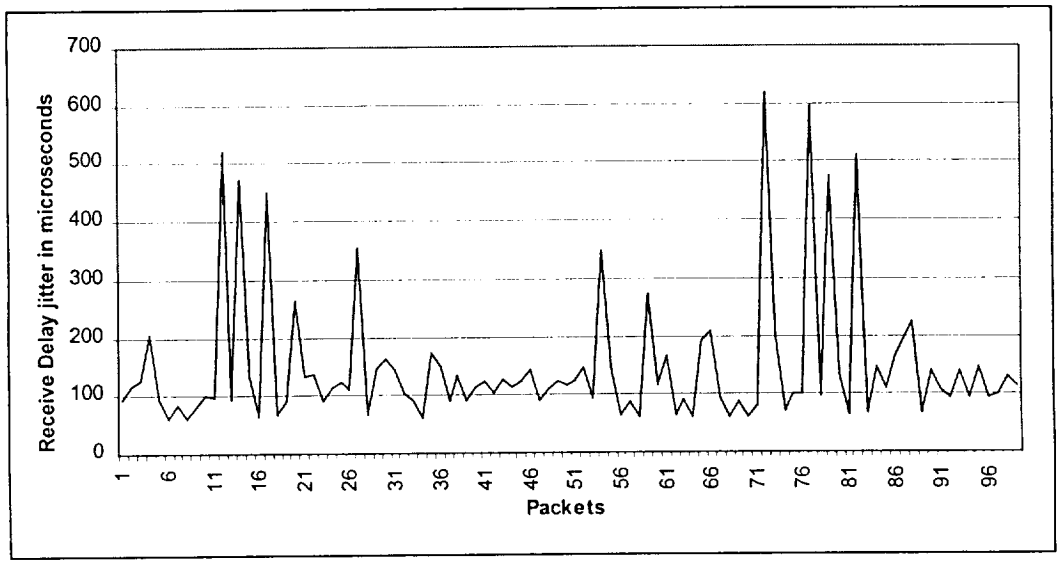


Figure 5.5. Receiving Delay Jitter Without Interruption From Packet Generator, Test 2

We found that CBQ provides desired bandwidth sharing. Figures 5.6 and 5.7 show packet-transfer delay for a bandwidth-sharing test. In this experiment, we suppose packets are from an external port and do not consume the CPU resources of the sending endpoint system. Two channels of the same priority each request 35% of the raw bandwidth. Since clocks at all endpoint systems are not synchronized, a packet transfer delay is calculated as the packet arrival time minus packet sending time with a modification of a constant value. The constant value is obtained through measurement and is equal to the system clock difference between two endpoint systems. We can see that the bandwidth is fairly shared between two sending channels at the sending host. Figure 5.8 represents the situation of two channels with different QoSs, with one requesting 50% of the bandwidth and the other requesting 20% of the bandwidth. Each channel gets its desired bandwidth share. As for the delay fluctuations in the low-bandwidth channel, they are caused by the scheduling algorithm itself. The figure also shows that the scheduler does not send out a packet every scheduling period so as to give fair bandwidth sharing among channels.

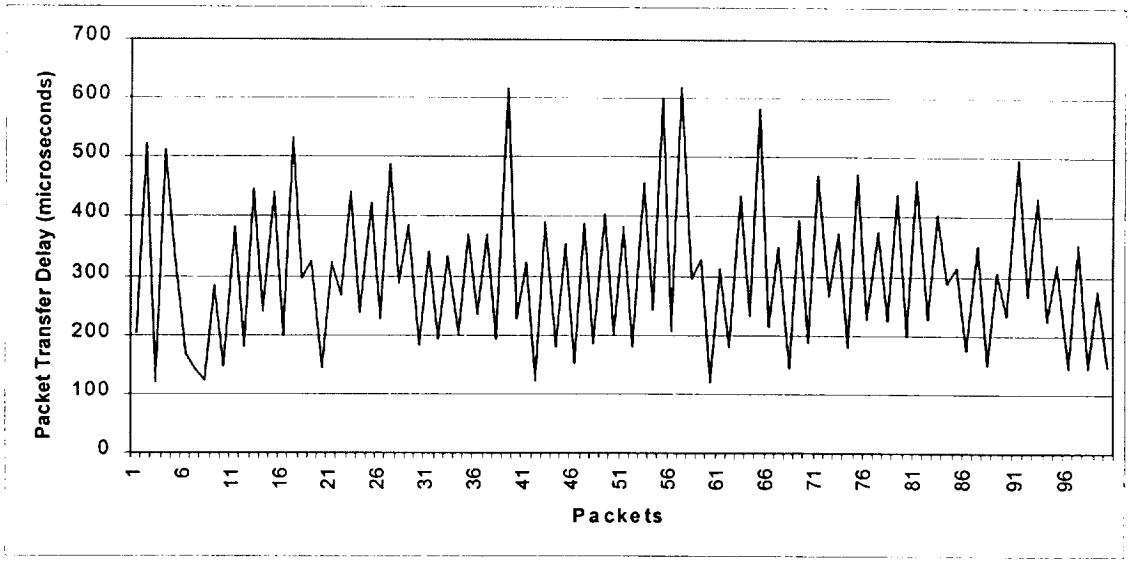


Figure 5.6. Packet Transfer Delay on Channel 1 – Bandwidth Sharing Test 1

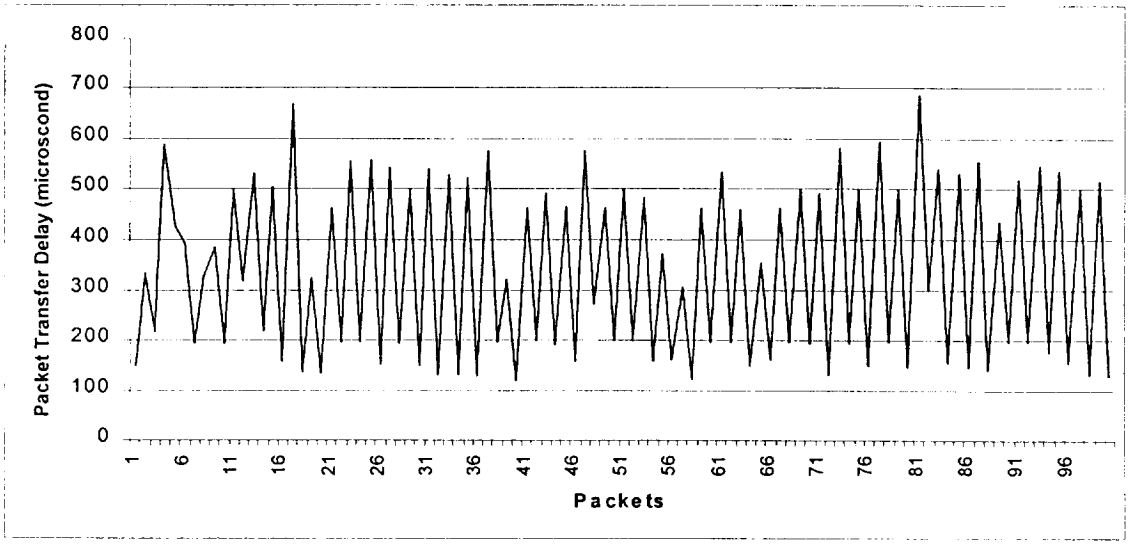


Figure 5.7. Packet Transfer Delay on Channel 2 – Bandwidth Sharing Test 1

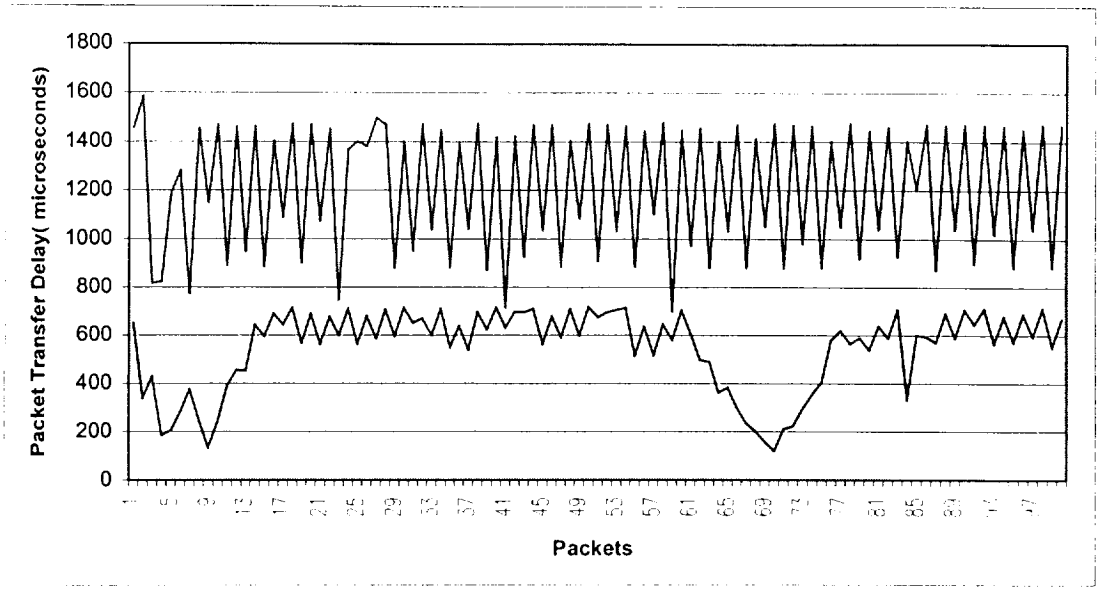


Figure 5.8. Packet Transfer Delay on Two Channels – Bandwidth Sharing Test 2

Figures 5.9 and 5.10 show end-to-end receiving delay in the situation in which a node has two receiving channels. Differing from the case of one node having two sending channels, our communication layer does not have an incoming packet scheduler since receiving a packet is passive. It is not necessary to introduce an incoming packet scheduler, which only adds overhead to the whole communication layer. In this test program, the packet receiver is running on a two-CPU symmetric multi-processing machine so as to greatly reduce effects that thread scheduling causes. The test program for receiver has two independent receiving threads to receive messages from two different channels. The two channels have the same priority and each ask for 50% of the bandwidth. The packet generation rate is 32 megabit per second. Our explanation of the difference between the two channels is the difference in system load at the two sending nodes.

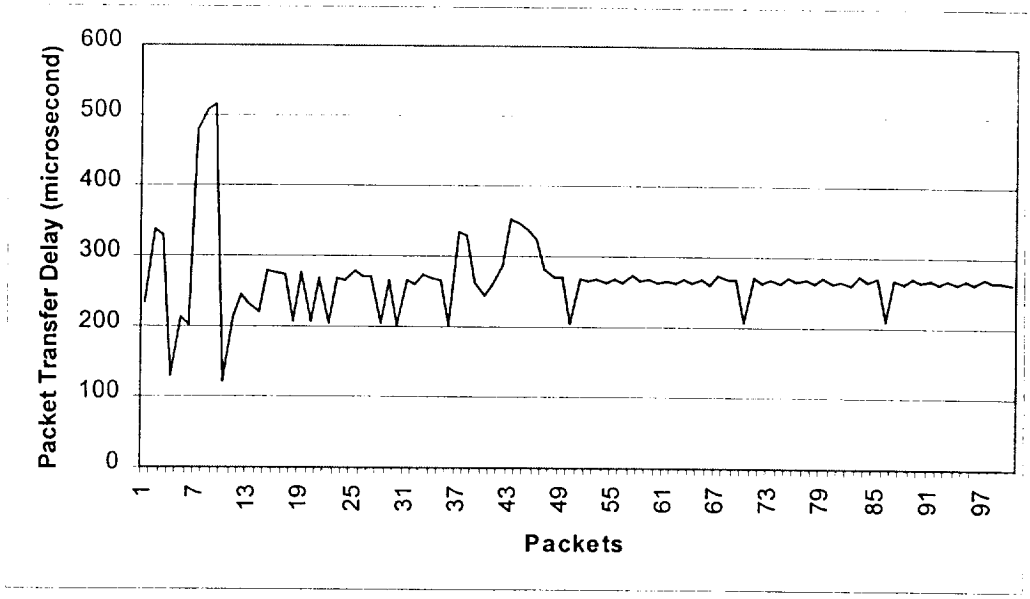


Figure 5.9. End-to-End Packet Receiving Delay on Channel 1 – One to Many Case

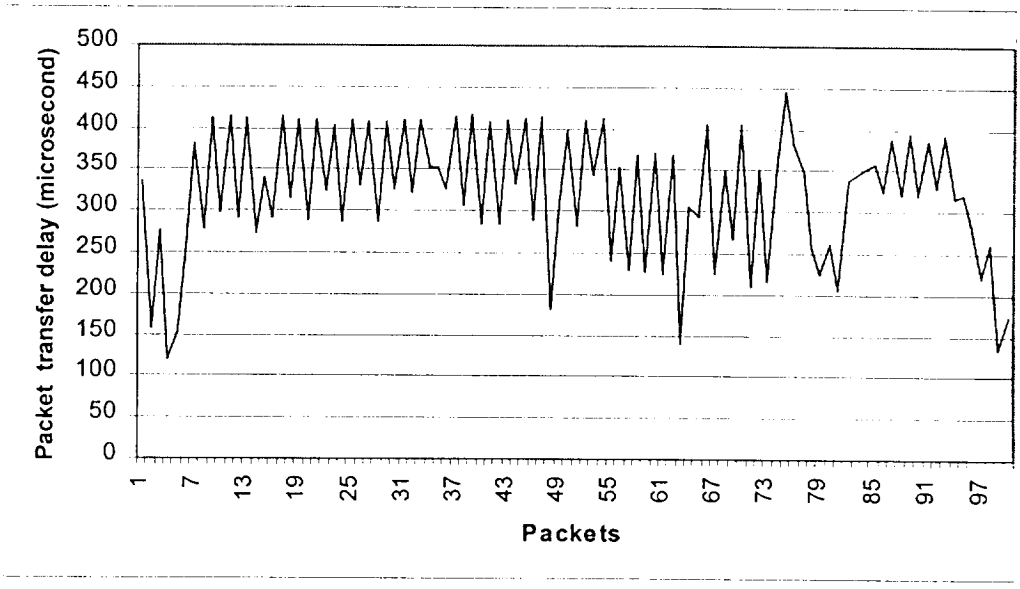


Figure 5.10. End-to-End Packet Receiving Delay on Channel 2 – One to Many Case

We tested our sub-network-wide admission-control mechanism under various conditions. It provides the expected behavior. For a new channel with an overage of QOS, global admission rejects this channel. When the accumulated bandwidth of all requested channels is larger than the capacity of a switch, we observed that the global admission test rejected the channel and resulted in admission failure. Sub-network-wide admission-control guarantees conflict-free access to the shared switches so that the switches can give bounded service time to each packet because queue overflow will never happen. The case where a low-priority channel is starved will never happen. Once a channel is admitted, the switch can always provide this channel with requested bandwidth.

Summary of Experiments

In the first experiment, we did not choose a high packet generation rate though network raw bandwidth can reach 100 megabit per second. That is because the packet generator and the packet scheduler sharing the CPU resources and multiple connections originating from the same node would make the scheduler unable to handle all incoming traffic. We also experimented with the effects of a queue overflow. A queue overflow will result in an indefinite packet transfer delay whose value depends on the queue size, packet arrival rate and channel bandwidth. For a given queue size and channel bandwidth, the faster the packets arrive, the more frequent the queue overflows. In the experiments, our data is sampled and is based on a small number of channel connections since they are sufficient for analyzing our communication layer. More complicated experiments were also conducted with equivalent observed results.

From the experimental results presented in this chapter, we can conclude that our communication middleware implementation is able to provide smooth end-to-end delay for a channel and the CBQ algorithm can provide fair bandwidth sharing if the underlying operating system can provide a guaranteed thread execution quantum. In other words, the middleware will be able to provide better performance in a real-time operating system. The experiments on this communication layer again confirm the research hypothesis.

Though middleware as an add-on component to the operating system cannot provide hard end-to-end QoS, our experiments provide some insights for QoS in a closed sub-network. Specifically, sub-network-wide bandwidth reservation mechanism can provide conflict-free access inside switches, which is a key point for QoS communication in a sub-network. Though RSVP on the switches can also realize bandwidth reservation, our method is much easier and more efficient when compared to RSVP method. Table 5.2 shows a summary of experimental results.

Table 5.2. Summary of Experimental Results

Experi-ment ID	Results	Significance of Results
Exp-1	Packet transfer delay can't be bound if packet scheduler can't get guaranteed CPU time.	It confirms the research hypothesis. Guaranteed CPU time for packet scheduler is indispensable to obtain hard end-to-end delay.
Exp-2	CBQ algorithm can provide expected bandwidth sharing and but not delay bound.	It also proves the research hypothesis, packet scheduling algorithm and associated QoS description will affect the extent that QoS can be achieved.
Exp-3	Sub-network-wide admission control can provide expected behavior.	Sub-network-wide admission control guarantee conflict-free access to the switches. Bound link-layer transfer delay becomes possible.

CHAPTER VI

CONCLUSION

In this chapter, we summarize the present work and present the conclusions that speak to the research problems. We also present the lessons we learned from this thesis study. Furthermore, future work and the implications of this study are also indicated.

Summary of Research Results

In this thesis, we made an in-depth analysis of various factors that affect QoS in packet-switching networks. Particularly, we focused on what endpoint systems can do for QoS communication in a closed high-speed sub-network in which switches provide only limited link-layer QoS, and lack sub-network-wide bandwidth reservation and dynamic session-based QoS support. In Chapter IV, we presented a sub-network-wide bandwidth reservation scheme that is a part of admission control in our implementation. Sub-network-wide admission control provides conflict free access to switches. Based on that, we designed and implemented a communication layer with QoS guarantee. Our experimental results on communication layer showed that hard end-to-end delay can be achieved if endpoint systems provide guaranteed CPU processing power for the packet scheduler. This proved the research hypothesis. That is, sub-network-wide bandwidth reservation and guaranteed CPU processing power at hosts for handling data traffic are both indispensable to achieving hard end-to-end quality of service. Our experimental

results on a class-based queuing algorithm also show that CBQ can provide percentage-based bandwidth sharing and its implementation is also relatively simple when compared to other queuing algorithms. But it cannot provide bound end-to-end packet transfer delay.

Experiments on IEEE 802.1p protocol showed that the protocol provides reasonable bandwidth sharing among connections with the same priority. However, its packet-transfer delay is not guaranteed at all for connections on a low-priority port. This means that priority-based packet scheduling is not suitable for QoS communication. Our analysis and experiments on RSVP showed that as a resource reservation protocol, RSVP is inefficient in closed, high-speed sub-networks. RSVP's receiver-initiated reservation strategy is not suitable for real-time distributed computing. In addition, the traffic model in RSVP is only suitable for continuous stream media, and using this model to describe non-periodic control messages will result in over-reserved resources. On the contrary, our communication layer provides a more efficient resource reservation method in which no refreshment is necessary and no hop-by-hop method is used for the admission test. Using RSVP for the hard QOS also requires guaranteed CPU time for the packet scheduler, which confirms the research hypothesis.

Lessons Learned

We learned several lessons from designing, implementing and experimenting with the communication layer. First we spent a large part of the time to track the system's behavior and tried to investigate various factors that cause the packet-transfer delay. The operating system scheduling heavily affects the packet-transfer delay.

Characterizing endpoint system behavior becomes necessary for precisely analyzing the time spent on individual activities. A simple fact is that it will take significant CPU time for fully using gigabit-per-second bandwidth, or else programmable protocol coprocessors must be introduced to offload the CPU. In other words, in a gigabit-per-second sub-network, endpoint system behavior will significantly affect network end-to-end packet transfer delay and jitter.

The second lesson learned during implementing and testing scheduling algorithms was that not only must the scheduling algorithm itself be simple, but implementation also needs to be highly efficient since scheduling algorithms will add overhead to message transfer. In a gigabit per second sub-network, the complexity of the algorithm may improve bandwidth usage or result in better fairness, but it usually consumes more CPU time, which in turn affects packet-transfer delay. That is a practical reason why most commercial gigabit per second switches do not provide dynamic session-based QoS. The packet scheduler would be better as a part of the operating system and more efficient transfer and control protocol should be used for improving performance. In particular, a real-time operating system is desirable on the endpoint system in order to obtain completely predictable behavior. A programmable Network Interface Controller (NIC) of sufficient speed could also help.

Future Work

This thesis is basically experimental research on QoS communication in high-speed packet-switching sub-networks. Several aspects of this research can be continued

in the study of QoS communication in high-performance embedded multi-computer systems (such as the Mercury RACE-Way system) (Mercury 1999).

Meta-computing is a hot research topic. Globus (Foster and Kesselman. 1997) is such a system that is based on MPI and TCP/IP. It organizes computing resources at different geographical locations into a meta-computer. A parallel application can access any CPU resources belonged to this meta-computer through using its G-MPI interface. However, currently it provides only limited QoS support. Obviously QoS support is necessary in such a system to achieve high performance and avoid communication and computing bottlenecks. Local and sub-network admission-control mechanism can be extended as node and sub-network resource management agents, respectively. By adding a global meta-computer-wide resource management agent, a three-layer resource management architecture could be set up to manage communication and CPU resources and improve overall performance. This could create a quality of service architecture instead of the current "sum-of-services" architecture.

Another interesting research field is to use our current communication layer as a tool to study the performance of different scheduling algorithms. Most performance analysis on packet-scheduling algorithms are theoretical or use a simulation method. Implementing different algorithms under same system environment and then comparing their performance, schedulability, and scalability would be significant contributions.

REFERENCES

Aras, C., J. Kurose, D. Reeves and H. Schulzrinne. 1994. Real-time communication in packet-switching networks. In *Proceedings of the Institution of Electrical Engineers* 82(1): 129-39.

Arvid, K. 1991. Protocols for distributed real-time systems. Ph.D dissertation. University of Massachusetts at Amherst.

Banerjea, A., D. Ferrari, B. Mah, M. Moran, D. Verma, and H. Zhang. 1996. The Tenet real-time protocol suite: design implementation, and experiences. *IEEE/ACM Transactions on Networking* 4(1):1-10.

Braden, R., L. Zhang, S. Berson, S. Herzog, and S. Jamin. 1997. Resource ReSerVation protocol (RSVP) -- Version 1 functional specification. Internet Engineering Task Force(IETF) RFC 2205.

Cilingiroglu, A., S. Lee, and A. Agrawala. 1997. *Real-Time communication*. College Park, MD: University of Maryland, Department of Computer Science. Technical Report UMIACS-TR-97-04.

Clark, D., S. Shenker, and L. Zhang. 1992. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. *ACM SIGCOMM Symposium on Communications Architectures and Protocols*: 14-26.

Crawley, E., L. Berger, S. Berson, F. Baker, M. Borden, and J. Krawczyk. 1998. A framework for integrated services and RSVP over ATM. RFC 2382. http://www.isi.edu/div7/rsvp/pub.html (Accessed 01 March 1999).

Cruz, R. 1991. A calculus for network delay, part I: Network elements in isolation. *IEEE Transactions of Information Theory* 37(1):114-21.

Demers, A., Keshav, S., and S. Shenker. 1989. Analysis and simulation of fair queuing algorithm. In *Proceedings of ACM SIGCOMM'89*, 1-12. Austin TX: ACM Publications.

Ferrari, D. and D. Verma. 1989. A scheme for real-time channel establishment in wide area networks. University of California at Berkeley, International Computer Science Institute. Technical report TR-89-036.

Ferrari, D. 1992. Design and applications of a delay jitter control scheme for packet switching internetworks. *Computer Communications* 15(6): 367-73.

Floyd, S. and V. Jacobson. 1995. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking* 3(4):365-86.

Foster, I. and C. Kesselman. 1997. Globus: a metacomputing infrastructure toolkit. *International Journal Supercomputer Applications* 11(2):115-128.

Golestani, S. J. 1990. Congestion-free transmission of real-time traffic in packet networks. In *Proceedings of IEEE INFOCOM*, 527-42. San Francisco, CA: IEEE Publications.

Ghanwani, A., J. W. Pace, and V. Srinivasan. 1997. A framework for providing integrated services over shared and switched LAN technologies. Internet Engineering Task Force (IETF). Internet Draft.

Hyman, J. M., A. A. Lazar and G. Pacifici. 1991. Real time scheduling with quality of service constraints. *IEEE Journal on Selected Areas in Communications* 9(7): 1052-63.

IETF. 1999. Integrated service mappings on IEEE 802 networks. http://search.ietf.org/internet-drafts/draft-ietf-issll-is802-svc-mapping-03.txt (Accessed 01 March 1999).

IEEE 1999. IEEE 802.1Q virtual bridge local area networks http://grouper.ieee.org/groups/802/1/vlan.html (Accessed 25 February 1999).

Kurose, J. 1993. Open issues and challenges in providing quality of service guarantees in high-speed networks. *ACM Computer Communications Review* 23(1):6-15.

Malcolm, N. and W. Zhao. 1995. Hard real-time communication in multiple-access networks. *Journal of Real-Time Systems* 8: 35-77.

McDysan, D. and D. Spohn. 1995. ATM theory and application. McGraw-Hill Series on Computer Communications.

Mercury Computer Systems Inc. 1999. The Race network architecture. http://www.mc.com/techlit/ (Accessed 25 January 1999).

Mizunuma, I., C. Shen and M. Takegaki. 1996. Middleware for distributed industrial real-time systems on ATM networks. A Mistsubishi Electric Research Laboratory. Technical report TR96-10a.

MPI Forum. 1994. Document for a standard message-passing interface. Knoxvile. TN: The University of Tennessee at Knoxville, Department of Computer Science. Technical Report CS-93-214.

MPI/RT Forum. 1998. Document for the real-time message passing interface(MPI/RT) standard. http://www.mpirt.org/ (Accessed 25 January 1999).

Parekh, A. 1992. A generalized processor sharing approach to flow control in integrated services networks. Ph.D dissertation, Massachusetts Institute of Technology.

Shen, C. 1996. On ATM support for distributed real-time Applications. http://www.merl.com/ (Accessed 25 January 1999).

Turner, J. S. 1986. New directions in communications. *IEEE Communications* 24(10): 8-15.

Wrege, D., E. Knightly, J. Liebeherr, and H. Zhang. 1996. Deterministic delay bounds for VBR video on packet-switching networks: fundamental limits and practical tradeoffs. *IEEE/ACM Transactions on Networking* 4(3):352-362.

Zhang, H. and D. Ferrari. 1993. Rate-control static-priority queuing. In *Proceedings of IEEE INFOCOM'93* 1:227-36. San Francicso, CA: IEEE Publications.

Zhang. L. 1991. Virtual Clock: A new traffic control algorithm for packet-switching networks. *ACM Transactions on Computer Systems* 9(2): 101-124.

Zheng, Q., K. Shin, and C. Shen. 1994. Real-time communication in ATM network In *Proceedings of 19th Annual Local Computer Network Conference*, 156-65. Minneapolis, MN: IEEE Publications.

APPENDIX

EXPERIMENTAL MEASUREMENTS

In this appendix, the data obtained in each experiment are listed. In each table, the first row represents the sequence numbers of sampled data packets. 1-26 means packet number 1 to packet number 26. Each column lists corresponding data values.

Table A.1. Data for Figure 3.4, End-to-End Packet Delay

Unit: μsec

1-26	27-52	53-78	79-104	105-130	131-156	157-182	183-208	209-234	235-256
101	104	41	48	189	103	107	100	101	95
156	88	198	205	136	89	88	85	90	86
46	101	46	46	44	98	95	100	126	99
137	87	83	134	180	84	85	88	134	86
43	95	102	45	43	102	99	99	45	98
79	85	85	137	82	87	88	86	139	86
42	107	93	46	104	100	103	102	49	103
146	89	83	84	86	89	88	86	84	88
47	99	99	104	43	97	93	158	103	99
476	85	90	87	141	85	85	141	87	87
56	96	102	102	50	107	106	45	93	95
146	83	87	88	139	87	88	208	86	87
47	103	93	97	44	102	104	47	103	103
81	88	83	85	135	88	87	133	87	86
42	102	101	99	42	97	95	46	106	104
81	87	89	86	137	83	83	139	86	87
44	97	104	102	97	102	101	44	97	97
147	84	85	88	86	86	88	83	83	84
45	103	96	97	105	105	108	105	102	102
208	88	84	88	85	89	88	89	87	87
103	102	100	102	101	98	95	100	97	95
91	85	89	86	87	84	82	88	87	
96	159	104	97	97	96	101	96	102	
130	203	87	84	85	86	87	85	87	
48	95	156	101	102	97	101	106	104	
83	90	142	89	86	89	87	88	86	

Table A.2. Data for Figure 3.5, End-to-End Packet Delay

Unit: μsec

1-26	27-52	53-78	79-104	105-130	131-156	157-182	183-208	209-234	235-256
51	145	2843	37	39	40	38	128	41	36
172	204	41	35	34	117	152	39	36	37
43	40	37	36	122	40	39	208	34	35
38	35	37	36	37	36	35	41	36	38
39	35	37	38	131	36	41	175	37	960765
36	35	38	1109924	39	35	36	131	280	49
37	115	3681	42	123	134	35	38	43	35
39	36	40	36	40	38	34	128	121	35
38	40	36	204	35	131	135	40	39	36
40	159	37	38	126	40	868	123	35	37
942	891	36	36	36	35	122	40	132	39
41	40	39	36	176	35	38	34	39	323014
37	36	3300	39	41	183	122	126	121	42
37	37	39	679	34	433	38	36	39	36
37	36	35	39	37	38	74	129	34	35
37	35	35	150	873	147	36	39	35	36
9151	37	36	129	178	39	71	34	36	37
101	1983985	38	134	85	125	38	35	1629770	39
80	47	3647	41	115	130	35	128	46	66108
37	38	38	36	39	39	38	685	37	39
35	38	35	36	73	35	34	38	38	36
37	156	36	140	36	35	34	35	39	
36	37	35	39	70	36	143	35	36	
37	39	37	124	37	35	35	151	40	
34	35	1877937	40	35	37	131	40	353353	
35	39	45	122	138	568	39	132	43	

Table A.3. Data for Figure 3.6., Packet Receiving Delay in Single RSVP flow

Unit: μsec

1-26	27-52	53-78	79-104	105-130	131-156	157-182	183-208	209-234	235-256
156	113	202	116	78	83	152	80	198	149
55	89	57	99	132	97	88	96	58	89
116	114	118	114	115	114	116	229	84	116
101	96	86	97	96	95	96	170	127	97
119	115	80	114	148	114	231	86	81	109
101	97	52	97	92	93	93	196	93	95
117	113	199	114	179	114	81	205	114	115
99	91	126	54	93	94	89	57	95	99
113	114	83	154	82	116	114	78	116	113
99	96	96	167	85	97	54	52	97	97
112	120	113	121	80	116	151	235	113	115
91	96	95	170	90	98	54	169	53	90
114	116	114	128	113	113	143	122	151	110
91	97	96	92	92	53	93	54	91	90
111	115	115	80	203	151	117	76	113	112
90	94	93	52	94	54	122	53	95	90
111	116	117	244	81	148	80	150	114	109
174	95	54	87	95	98	92	88	97	89
124	115	114	81	114	115	114	235	115	113
98	90	90	94	89	100	93	93	95	184
198	115	114	114	112	252	113	80	111	125
100	176	97	94	97	57	54	91	95	
119	127	115	113	114	119	150	109	112	
90	92	97	94	101	98	54	54	53	
81	81	114	112	211	79	265	148	143	
89	172	94	52	92	52	93	170	55	

Table A.4. Data for Figure 3.7, Packet Receiving Delay in Multiple RSVP flows

Unit: μsec

1-26	27-52	53-78	79-104	105-130	131-156	157-182	183-208	209-234	235-256
141	181	138	89	94	128	86	136	320	88
109	136	67	58	110	147	111	63	66	106
88	136	88	86	128	178	126	169	88	91
59	103	114	256	151	105	60	108	105	115
214	89	188	180	135	91	137	171	127	133
114	232	63	102	105	137	107	104	61	106
167	95	268	90	88	90	266	90	87	177
105	150	108	105	63	61	98	146	153	228
87	89	89	89	137	228	88	89	174	179
112	60	116	59	64	230	147	61	107	63
188	85	124	85	344	135	90	133	89	87
64	153	61	150	100	64	61	108	61	60
269	167	133	175	91	86	266	173	167	273
110	108	66	166	148	58	112	103	63	137
170	129	172	171	88	85	253	93	359	229
107	60	144	66	59	250	70	61	150	62
135	129	262	89	135	181	240	270	134	90
197	110	68	60	106	106	68	65	62	59
137	168	130	85	165	93	140	181	86	185
108	140	64	156	137	146	106	200	58	100
89	131	86	170	90	90	137	171	241	170
150	203	59	103	147	59	110	63	65	
87	92	168	173	90	86	92	87	271	
110	106	205	104	117	308	155	59	107	
222	128	177	91	137	172	91	87	172	
65	193	63	202	63	63	108	107	63	

Table A.5. Data for Figure 5.2, Packet Receiving Delay Jitter

Unit: μsec

1-20	21-40	41-60	61-80	81-99
515	91	61	222	107
96	61	81	66	169
480	81	400	81	138
92	475	117241	358819	93
371	268463	107	89	200
135	216	58	518	139
63	110	44461	92	106
82	129	75362	479	94
425	205	173	190	61
89	179	66	66	170
406	67	86	390	66
287138	88	58	65	85
183	61	44233	87	62
108	324	195576	57	83
92	67	182	198243	565
62	89	108	281	117462
166	112	91	67	103
64	188	61	89	60
90	66	88	61	43949
149	86	115	137	

Table A.6. Data for Figure 5.4, Receiving Delay Jitter

Unit: μsec

1-20	21-40	41-60	61-80	81-99
93	132	124	165	64
116	138	105	65	510
126	92	127	91	69
205	113	114	63	145
94	123	123	191	112
61	111	143	209	167
85	355	92	96	198
61	69	112	62	226
80	148	123	87	69
308501	163	116	63	141
99	144	125	81	106
522	104	145	619	93
96	92	93	7403	139
473	63	347	71	93
135	172	142	303251	145
64	150	66	102	93
450	92	87	596	102
68	134	61	98	130
91	92	273	475	115
264	115	116	135	

Table A.7. Data for Figure 5.6, Packet Transfer Delay

Unit: μsec

1-20	21-40	41-60	61-80	81-100
204	323	322	312	462
523	268	122	181	228
120	441	390	435	403
510	238	181	233	289
337	423	355	583	315
168	228	151	215	176
142	488	387	348	352
122	288	187	144	149
283	385	404	393	308
146	184	203	188	233
383	341	383	469	495
181	195	181	267	267
447	334	457	371	432
242	203	244	180	226
440	369	601	473	320
200	235	208	229	143
532	369	620	374	354
295	194	297	225	144
324	616	329	439	274
144	227	121	199	146

Table A.8. Data for Figure 5.7, Packet Transfer Delay

Unit: μsec

1-20	21-40	41-60	61-80	81-100
151	460	460	533	686
332	199	200	197	304
220	555	492	459	539
586	198	191	149	157
425	556	464	355	529
393	153	160	162	146
194	541	575	462	553
328	194	273	199	142
385	499	462	497	435
195	149	201	196	199
497	538	496	492	518
318	131	201	131	199
530	526	482	580	545
218	133	159	195	177
503	522	371	497	537
159	128	162	149	157
667	575	307	593	499
138	198	124	198	133
325	321	460	496	516
136	120	197	148	128

Table A.9. Data for Figure 5.8, Packet Transfer Delay

Unit: μsec

Upper curve:

1-10	11-20	21-30	31-40	41-50	51-60	61-70	71-80	81-90	91-100
1458	892	1074	953	715	910	975	879	1041	900
1585	1462	1455	1472	1423	1471	1456	1469	1460	1473
818	949	747	1040	928	1034	882	982	928	1021
824	1464	1368	1449	1470	1466	1402	1466	1402	1467
1190	887	1402	880	1036	885	1034	883	1203	883
1281	1403	1381	1396	1470	1445	1470	1401	1471	1450
774	1090	1497	1043	886	1103	882	1051	873	1039
1455	1474	1473	1476	1405	1479	1413	1477	1469	1473
1151	902	881	871	1084	698	1052	920	1040	884
1470	1471	1400	1420	1475	1447	1476	1445	1472	1467

Down Curve:

1-10	11-20	21-30	31-40	41-50	51-60	61-70	71-80	81-90	91-100
650	391	563	652	631	675	606	211	638	645
340	455	676	670	697	695	498	223	589	711
429	453	599	600	695	705	489	295	707	564
186	644	710	710	710	713	362	353	333	679
205	595	564	549	563	515	384	406	601	571
281	690	679	638	678	636	299	580	593	689
377	644	587	541	591	517	236	619	571	594
249	714	707	695	709	646	202	565	691	712
134	567	596	624	599	580	157	589	589	551
246	689	713	716	717	703	120	539	707	667

Table A.10. Data for Figure 5.9, End-to-End Packet Receiving Delay

Unit: μsec

1-20	21-40	41-60	61-80	81-99
233	269	260	267	266
338	206	286	264	261
329	269	353	269	273
129	265	347	263	267
213	278	336	268	270
203	271	324	261	211
479	271	282	274	268
509	204	271	268	264
516	266	272	268	270
120	203	205	209	265
213	267	269	270	268
245	261	265	264	263
232	274	269	269	269
222	268	264	263	264
280	266	268	271	269
276	202	262	266	264
275	334	273	269	270
209	328	267	262	267
276	264	268	270	267
207	245	262	264	262

Table A.11. Data for Figure 5.10, End-to-End Packet Receiving Delay

Unit : μsec

1-20	21-40	41-60	61-80	81-99
336	411	408	225	207
157	325	285	368	337
276	404	411	140	345
120	287	333	304	351
154	411	412	294	357
257	331	290	405	324
381	407	414	226	388
279	287	182	348	321
412	407	308	267	395
298	327	396	406	320
415	409	282	210	385
292	322	411	350	328
412	409	344	218	392
275	352	412	360	316
341	353	242	446	320
292	327	353	382	282
415	414	231	348	222
316	308	369	256	260
410	416	227	226	134
290	285	371	260	173