Spring 1997

Designing a High-Quality Network: An Application-Oriented Approach

Sudheer Dharanikota
Old Dominion University

DESIGNING A HIGH QUALITY NETWORK: AN APPLICATION-ORIENTED APPROACH

by

Sudheer Dharanikota
Indian Institute of Science, Bangalore, India,
Nagarjuna University, Vijayawada, India

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements of the Degree of

DOCTOR OF PHILOSOPHY

COMPUTER SCIENCE

OLD DOMINION UNIVERSITY
May 1997

Approved by:

Kurt J. Maly (Director)

C. Michael Overstreet (Member)

David E. Keyes (Member)

Wayne H. Bryant (Member)

ABSTRACT

DESIGNING A HIGH QUALITY NETWORK:
AN APPLICATION-ORIENTED APPROACH

Sudheer Dharanikota
Old Dominion University, 1996
Director: Dr. Kurt J. Maly

As new computer network technologies emerge, the application designers and the application users expect an increasing level of quality of service from them. Hence, it is a common practice in the newer technologies to provide more Quality of Service (QoS) components. Until now, these QoS solutions have been both network-technology specific and network-oriented solutions. In this thesis, we present an application-oriented approach to design a high quality network which is independent of the underlying communication technology. In this thesis, we propose a QoS architecture to "provide predictable performance to the end-to-end application users in a high quality networking environment." In our architecture, QUANTA (Quality of Service Architecture for Native TCP/IP over ATM networks), we integrate different application requirements and different existing native QoS architectures into a single end-to-end architecture.

Through experimentation we identify the architectural issues and the different QoS components required. We propose solutions which include isolation of the applications and managing the knowledge of the applications. The issue of isolating an application is subdivided into classification and identification of the applications. In addressing these issues we propose a ripple-through classification mechanism and

a Generic Soft State (GSS) identification mechanism. To manage the knowledge of the applications we propose different QoS components, such as a GSS negotiation mechanism, a GSS communication mechanism and a GSS monitoring mechanism (such as GSS Relays and GSS Agents).

QUANTA's overhead is measured by running applications with different life-times and QoS requirements with and without QUANTA. For a transaction-oriented applications, the overhead induced by using QUANTA is larger than the benefit of using QUANTA. In high data rate applications and in long life time applications, a predictable performance to the applications is achieved with very low overhead by QUANTA. We demonstrate that QUANTA can manage and maintain Quality of Service for different classes of applications under varying host and network load conditions transparent to the application user. Using QUANTA, we can reach nearly 80% of the channel utilization under loaded conditions, whereas without QUANTA, the load on the network can reduce the channel utilization to 40%. Quanta has reduced a 350 msec delay under loaded conditions to less than a 10 msec delay. With the exception of short transition periods, QUANTA can sustain throughput to within the bounds of the specification of the user. We identify the limitations of QUANTA as currently proposed and discuss possible enhancements to it and other such architectures to remedy these limitations.

To

Mother, Father, Kishore and Nadima

ACKNOWLEDGMENTS

My sincere acknowledgments to Prof. Maly for his encouragement, time, training and suggestions in the capacity of an advisor; for his support and understanding of the needs of a graduate student in the capacity of the chairman of the department; and most importantly for his patience, the leniency he has given to me, and his coping with my idiosyncrasy like a friend. I have tried to inherit many of his qualities, such as organizing my thoughts and time, handling tough situations, and thinking quickly. I am lucky to have such an advisor.

I thank my committee members Dr. David Keyes, Dr. Mike Overstreet, and Dr. Wayne Bryant for their interest in my work and their suggestions from my candidacy until defense. My acknowledgments to Dr. Zubair for his *to-the-point* questions which gave me more confidence in my work as I struggled to answer them.

One person in the department seemed very serious and reclusive when I first joined. But in reality, when I became close to him, I understood his grace, his helpful nature and his friendliness. I am glad that I could recognize his personality at the earlier stages of my stay at ODU. He was with me when I needed any technical or personnel suggestions. He provided me with all the equipment needed to complete my work, and let me borrow his three little angels when I wanted to take my mind off the work. Thank you Ajay.

Staying so far away from home — from the people I love and from the places I adore — made me always think about the necessity of doing a Ph.D. Three people in my life take credit not only for prompting and encouraging me to pursue my life ambition but also being available to me when I became emotional. I love them for being what they are, and am proud of what they are as they are proud of me. Thank you Mom, Dad and Kishore for every good event happening in my life — you are

the reason for it.

I am thankful to one person who made two state transitions in my life during my stay at ODU - one from an enemy to a friend and the other from a friend to my wife. She still plays the role of these three characters successfully. She is my chalk-board of ideas, eternal sink of my emotions, and my soul (and of course my banker). I am indebted to her for the endless walks we had as enemies (quarreling), as friends (chatting over the past experiences), and as partners (planning everything). It is impossible for me even to imagine finishing my Ph.D without her round-the-clock attention. Thank you Naddo.

I want to acknowledge the host of friends I made at ODU and the quality time we had together. Thank you Caleb and Loretta Cutherals for being my host parents, introducing me to the American culture, and being with me all these four years whenever I needed a break. I will always be in touch with you, the confused V (Venkat) and the chaotic V (Vittal). Thank you Roy for being my first roomie, my best friend, and of course for the endless cooking adventures we had (deep frying the eggs!). I will always remember you Kumar for the fun times we had, the coast-to-coast trip we made, and the emotional trauma you created when you tried to ignore me. Rajesh, do you remember the long fight we had before becoming such close friends? Thank you Basu Vasu for the unending discussions we had on spiritual and philosophical topics. Rai Bhaskaram, it is impossible to get your organization and determination; I will always admire you. Thank you Pandyaji, Archita, Abhay and Ditta for being my accomplices on the elaborate singing sessions had in Gandhi Nagar (Bolling Square). Thank you Usha for all your birthday cakes, Hima for the Pulihara, and Ranjita for the Mysorepaks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Prof. Fouad Tobagi quoted at High Performance Networking Conference 1995, "The application is the King! ... Networking is today driven by the applications."

As Prof. Tobagi states, the networking community* understands that the end-system application user needs to be satisfied. In this thesis, we propose an architecture to better meet the user requirements with an existing set of resources.

In the following section, we will discuss the motivation for the problem at hand, define the scope of the problem, outline of the issues involved in this problem, and survey related-work.

1 Motivation

When ISDN (Integrated Services Digital Network) technology was initiated in the early '80's, the major issue was integration (Figure I.1) of different services such

*This dissertation is prepared conforming to the journal paper submission guidelines of the ACM SIGCOMM proceedings, 1993 and IEEE Transactions, 1996.

In this chapter, the terms we and us refer to the networking community, unless it is implicit that it is the author.

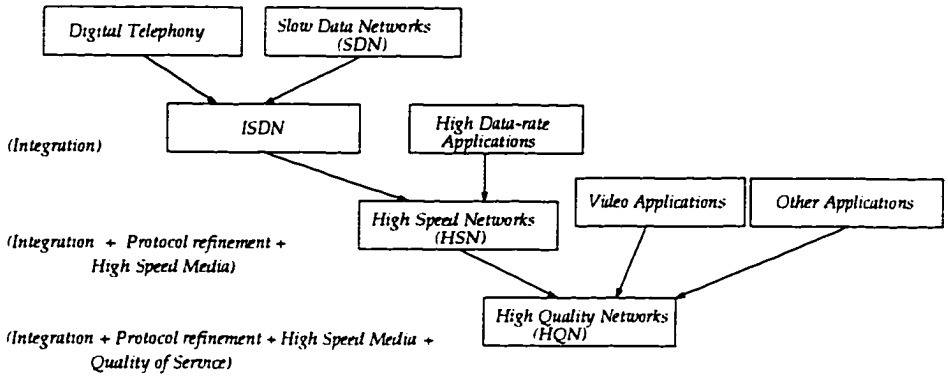


Figure I.1: Network technologies and their driving issues

as digital telephony, and slow data networks (using, e.g., X.25). With the development of LAN and MAN technologies, and High Data-rate Applications (such as LAN-interconnection), we moved into the era of High Speed Networks (HSNs). In HSNs, the major issues are the development of high bandwidth media (e.g., fiber optics), and the refinement of protocols to use the high bandwidth media (e.g., Frame Relay). With abundant bandwidth newer applications such as World Wide Web (WWW) were developed and also started integrating video applications, such as video collaborative applications, onto the network. With such intelligent applications the user of the applications may now need to request a specific Quality of Service (QoS) from the communication subsystem, which in this thesis means the end-system protocol stack and the network connecting the end-systems. This led to the design of the High Quality Networks (HQNs). Standards committees such as the ATM forum and the IETF (Internet Engineering Task Force) are making progress in providing QoS guarantees inside the network. To utilize the QoS facilities provided by the network protocols and to interwork with the above native QoS architectures we need an end-user-to-end-user QoS architecture. The emphasis in this thesis is

to discuss the issues addressed by such architectures and to design and develop a prototype of one such QoS architecture.

It is noteworthy that fundamental issues are still the same irrespective of the change in the technology. The common issues that are addressed in all these technologies are:

- Addressing and binding;
- Routing;
- Controlling data flow, such as congestion schemes, and flow control schemes; and
- Error handling, such as detection, correction, and retransmission.

Though fundamental issues did not change, the intensity with which these issues are addressed have changed from one technology to the other. For example, in Figure I.2 we present a relationship between the emerging technology and the degree to which the issue of error handling is addressed in that technology.

In slow data networks (SDNs), because of the low latency $\times$ bandwidth product, the error handling can be done with the help of slow-reaction mechanisms such as window-based retransmission. In digital telephony no loss of data should occur. If the losses occur, recovering from losses will not help such applications. Hence, when a technology is developed to support digital telephony, measures are taken to prevent losses. When SDNs and digital telephony are integrated in ISDN, error handling should be emphasized to cover slow-data networks. When ISDN is integrated with high data-rate applications to develop HSN technology, the importance given to error handling should be higher (because of the high latency $\times$ bandwidth product of such technologies) than that in SDNs, as at any given time the amount

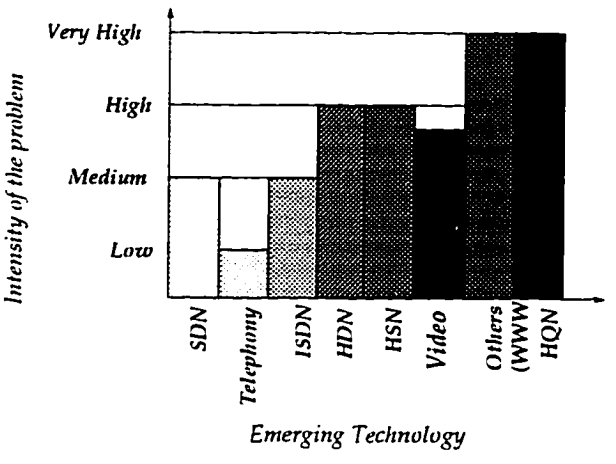


Figure I.2: Intensity of error handling *versus* emerging technology

of data in transit is proportional to its bandwidth. Hence, in such technologies, error handling mechanisms such as window-based retransmissions are inadequate. By integrating applications such as video collaborative applications with bounded error requirements, the issue of errors should be handled meticulously to provide tighter control on losses and errors. Therefore, when we propose an architecture for newer technology (such as HQNs), we should revisit the fundamental issues and suggest appropriate solutions to them.

Conventional network protocols cannot be used in HQNs because of the following reasons:

i. In conventional connectionless network protocols (such as IP), there is no identity to an application in any form. In such a scenario, multiplexing applications with different QoS requirements can encounter unpredictable QoS behavior. For example, multiplexing a high data rate application with a time critical application may lead to higher delays to the later application and may also lead to unpredictable losses.

ii. Also, in conventional network protocols, knowledge of the application is not maintained. Hence the network cannot react as the application anticipated at the time of resource quench.

Hence it is not possible to provide a QoS guarantee in the existing network protocols. This will require us to define additional issues necessary for providing QoS guarantees in addition to our earlier set of issues. We outline these additional issues involved in attaining a QoS architecture in the next section.

2 Outline of the issues

As mentioned in the previous section, the fundamental issues an HQN should address are related to the isolation of an application and manage and maintain the QoS knowledge of that application. Apart from these fundamental questions, proposed solutions should also satisfy some basic requirements in order to be scalable to larger networks and a larger set of application's QoS requirements. These issues can be itemized into different groups as follows:

i. Isolation of the application

   a. Classification of applications: This helps in isolating an application or a group of applications from each other. In combination with different scheduling schemes, a classification mechanism can provide QoS guarantees in a connectionless network protocol suite.

   b. Identification of the application: An application in HQN is identified, apart from the regular connection identifier used in the earlier network protocols, by its QoS requirements. Thus, in future networks an ap-

plication should be identified by the tuple <Connection identifier, QoS identifier>.

ii. Knowledge of the application QoS requirements

a. Specification and Translation of application QoS: Applications require a simple and user-friendly QoS specification mechanism to specify their QoS requirements in its own terminology. Then a QoS translating mechanism is invoked to translate these QoS requirements, without losing any information, into the host and the network understandable QoS parameters.

b. Communicating application characteristics: To propagate the application requirements, to reserve required resources for this application and also to relinquish them once the connection is terminated, in HQNs, a QoS communication protocol needs to be devised.

c. Negotiation of QoS: To adjust the applications QoS to the dynamics of the network, a negotiation mechanism is required between different resource managers in the application communication path.

d. QoS provision: QoS provision algorithms are required to guarantee the agreed upon QoS of an application. Furthermore, a QoS manager is required to dynamically manage and monitor different applications QoS, during the connection establishment phase in allocating the resources, and during the data transfer phase to monitor the application (to check if it is within the requested QoS) and the network resources (to dynamically modify different resources allocated to the application to maintain the promised QoS).

iii. Other issues related to the solution

a. Fairness in allocating resources is important when connections share a pool of resources (such as in TCP/UDP) and when several connections are multiplexed into a single connection (such as in IP).

b. Scalability of solutions to accommodate the expansion of the networks and the increase in the range of the applications QoS requirements is a major issue in HQNs.

c. Interoperability with the existing protocol suites and smooth transition to the next generation of the protocols is an important issue for the success of a solution.

d. Efficiency of the solution in terms of the other existing proposals and the cost of the solution in terms of the amount of additional work a protocol need to perform for this solution will decide the overhead of the solution.

Apart from these issues, the fundamental issues as mentioned in the section 1.1 are carried into HQNs. But in this thesis, we limit ourselves to address the QoS architectural issues as outlined in this section.

3 Problem and objective definition

In this thesis, we propose a QoS architecture to "provide predictable performance to the end-to-end application users in a high quality networking environment."

All applications, from complex distributed applications such as "Video Collaborative (VC)," to simple point-to-point applications such as "ftp," expect pre-

dictable performance. The terms predictability and performance need explanation in the context of this work. An application behaves predictably when the application user observes the statistical behavior of the application and it is consistent with the requested behavior. The term "performance" is a relative term, which is a measure of the behavior of an application. Thus, an application is said to be performing predictably when it is in the behavioral range as anticipated by the user. The following paragraphs give an example to explain these concepts.

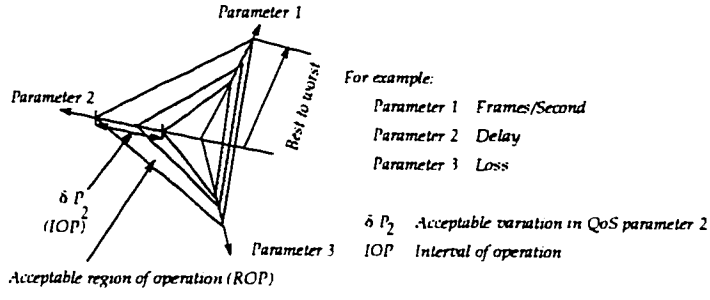


Figure I.3: Acceptable region of operation (ROP) for an application

Consider a Video Collaborative (VC), such as an IRI (Interactive Remote Instruction) [1] application, in which audio and video are sent across the network among a group of collaborative users (e.g. between a teacher and students of the class). A user in such an environment requires a set of performance measures (Quality of Service parameters) to be met to use this application. For example, a user may be interested in the number of frames received per second (FPS), latency, or delay between the sender and the receiver, for a good on-line interaction, and less loss and jitter in terms of user perception. These requirements define an Interval Of Operation (IOP) for the application as shown using a Kiviat diagram in Figure I.3. Each axis in the Kiviat diagram represents a QoS parameter. The QoS marking on these axis range from the best quality, being the innermost point, to the lowest quality, being the outermost point. As shown in Figure I.3, these acceptable deviations

create a Region Of Operation (ROP). In the diagram the innermost triangle represents the best quality the application would expect. This is limited by factors such as the minimum delay, which is introduced by the protocol suite processing overhead and the propagation delay of the signal. The outermost triangle represents the worst performance an application would accept. We can say that the application is performing predictably if its performance measures are within ROP, as defined by the user. Deviations in IOP manifest themselves differently in different applications: in a VC application this may be reflected as a reduction in the number of frames per second (FPS), and in an ftp application, this may be reflected as a reduction in the throughput.

The degradation in the application performance is mainly due to three causes: the host load condition, end-system protocol behavior in high speed networks (HSN), and the network load condition [2]. Kernel-to-user copying, buffer availability in the kernel, CPU occupancy, number of interrupts and speed of the user-network interface are examples of host-related issues. The end-system protocol suite's dynamic behavior is a major contributor to the variation in performance of an application [2, 3]. A clear understanding of the controlling parameters in the end-systems is necessary to obtain predictable performance. The behavior of the network at different load-conditions will influence the application's statistical behavior.

Since QoS of an application is influenced by the end-user to the end-user components, the QoS architecture also needs to address the issues mentioned in section 1.3 from the end-user to the end-user perspective. This thesis provides one such architecture to provide predictable performance to the applications.

4 Survey of related work

This section provides the survey of work within the scope of this thesis. We first discuss the research that is the precursor in the area of the QoS provision and then summarize the other current related work. We defer the discussion on the above discussed issues to the issues chapter (Chapter III), where we compare the existing approaches with ours.

4.1 Survey of QoS provision components

As an initial attempt in transition from HSNs to HQNs, many researchers evaluated the existing protocols with certain high speed extensions. They identified the protocol control parameters and their inter-relationships to obtain QoS. In the process, basic resource negotiation protocols were developed, and the existing hooks in the conventional protocols were used, to the limit, to upgrade them to support HQNs. As a next setup towards HQNs, to accommodate the current and future dynamics of the network, resource allocation and feedback control algorithms were studied. In the following paragraphs, we summarize the work performed in these areas, which is relevant to this thesis.

High Speed extensions

RFC1323 [4] presents a set of TCP [5] extensions to improve performance over large bandwidth × delay product paths and to provide reliable operation over very high-speed paths. It defines new TCP options for scaled windows and time stamps, which are designed to provide compatible interworking with TCP's that do not implement the extensions. The time stamps are used for two distinct mech-

anisms: RTTM (Round Trip Time Measurement) and PAWS (Protect Against Wrapped Sequences). Selective acknowledgments are not included in [4].

RFC1152 [6] reports on the gathering of a small renowned people in the area of very-high-speed networking group in April 1990. David Clark (from MIT) claimed that existing protocols would be sufficient to go at a Gigabit per second, if that were the only goal. Van Jacobson (from LBL) reported results that suggest that existing protocols can operate at high speeds without the need for outboard processors. He also argued that resource reservation can be integrated into a connectionless protocol such as IP without losing the essence of the connectionless architecture. This is in contrast to a more commonly held belief that full connection setup will be necessary in order to support resource reservation. Jacobson, during this meeting, said that he has an experimental IP gateway that supports resource reservation for specific packet sequences. Dave Borman, from Cray Research, described high-speed execution of TCP on a Cray where the overhead is most probably the system and I/O architecture rather than the protocol. He believes that protocols such as TCP would be suitable for high-speed operation if the windows and sequence spaces were large enough.

RFC1077 [7] is the report of the Gigabit Networking (GN) group workshop. The critical question addressed in this RFC is how the fiber-optic provided raw bandwidth can be used to satisfy the requirements identified in the workshop: (1) provide bandwidth on the order of several Gbit/s to individual users, and (2) provide modest bandwidth on the order of several Mbit/s to a large number of users in a cost-effective manner through the aggregation of their traffic. The participants of this workshop feel that the next-generation architecture has to be, first and foremost, a management architecture. The directions in link speeds, processor speeds and

memory solve the performance problems for many communication situations so well that manageability becomes the predominant concern.

The gigabit networks will need to take advantage of a multitude of different and heterogeneous networks, all of high speed. In addition to networks based on the technology of the GN, there will be high-speed LANs. A key issue in the development of the GN will be the development of a strategy for interconnecting such networks to provide gigabit service on an end-to-end basis. This will involve techniques for switching, interfacing, and **management coupled with an architecture that allows the GN to take full advantage of the performance of the various high-speed networks** [emphasis added].

They state that all of the information that an analyst would consider crucial in diagnosing system performance is carefully hidden from adjacent layers. One "solution" often discussed, but rarely implemented, is to condense all of this information into a few bits of "Type of Service" or "Quality of Service" request flowing in **one direction only from application to network.** It seems likely that this approach cannot succeed, both because it applies too much compression to the knowledge available and because it does not provide two-way flow, which is centrally considered in this thesis.

Research in control parameters

RFC1106 [8] discusses two extensions to the TCP protocol to provide a more efficient operation over a network with a high bandwidth × delay product.

The two options implemented and discussed in this RFC are:

1. Negative acknowledgments (NAKs): This extension allows the receiver

of data to inform the sender that a packet of data was not received and needs to be resent. This option proves to be useful over any network path (both high and low bandwidth × delay type networks) that experiences periodic errors such as lost packets, noisy links, or dropped packets due to congestion. The information conveyed by this option is advisory and if ignored does not have any effect on TCP whatsoever.

2. Big Windows: This option will give a method of expanding the current 16-bit (64 Kbytes) TCP window to 32-bits of which 30 bits (over 1 Gigabytes) are allowed for the receive window. No changes are required to the standard TCP header. The 16 bit field in the TCP header that is used to convey the receive window will remain unchanged. The 32 bit receive window is achieved through the use of an option that contains the upper half of the window. This option is necessary to fill large data pipes such as a satellite link.

RFC1110 [9] says that TCP Big Window option discussed in RFC 1106 will not work properly in an Internet environment, which has both a high bandwidth × delay product and the possibility of Dis ordering and duplicating packets. In such networks, the window size must not be increased without a similar increase in the sequence number space. Therefore, a different approach to big windows should be taken in the Internet.

RFC1072 [10] says that recent work on TCP performance has shown that TCP can work well over a variety of Internet paths. However, a fundamental TCP performance bottleneck for one transmission regime still exists: paths with high bandwidth and long round-trip delays. Clever algorithms alone will not give us good TCP performance over these "long, fat pipes;" it will be necessary to actually extend the protocol. This RFC proposes a set of TCP extensions for this purpose.

They propose extension of window size, sending cumulative acknowledgments and provide reliable RTT (Round Trip Time) measurement algorithms.

Control parameter inter-relationships

Sudheer et al., [2, 3] investigated the limitations of the normal implementation of TCP(UDP)/IP and described an application-oriented analysis in high-speed Local Area Networks, such as ATMs.

They conducted tests to measure aberration in Quality of Service of an application in terms of connection establishment time, throughput, and loss with respect to block size. They report the effect of TCP window size and the Silly Window Syndrome (SWS). Suggestions are made to avoid the SWS and effectively control TCP window size to increase throughput. Data losses of nearly 27% are observed with a UDP application. Knowledge of the status of the network can be used effectively by a host to reduce losses. They demonstrate this point with the help of a simple rate control algorithm at the user level in the UDP/IP environment. Results obtained from the above experiments are used to analyze a simulated Distributed Computing application.

D. E. Comer [11] reported that TCP buffer sizes and the ATM interface maximum transmission unit have a dramatic impact on throughput. They observed a throughput anomaly in which increase in the receiver's buffer size decreases throughput substantially. They explain the anomalous behavior and describe a solution that prevents it from occurring, and they analyze the performance of TCP/IP with respect to buffer sizes.

Jacobson [12] presented the reasons behind the congestion collapses and the

involvement of timers and the improvement. He discussed seven new timer algorithms implemented in the 4BSD TCP, namely: (1) round-trip-time variance estimation, (2) exponential retransmit timer backoff, (3) slow-start (4) more aggressive receiver acknowledgment policy, (5) dynamic window sizing on congestion, (6) Karn's clamped retransmit backoff, and (7) fast retransmit. He claims that measurements and the reports of beta testers suggest that the final product is fairly good at dealing with congested conditions on the Internet. Reference [12] is a brief description of (1) - (5) and the rationale behind them. This will be the basic work to understand the behavior of various timer algorithms.

QoS negotiation algorithms

Once a contract is agreed upon between the user and the provider, the provider has to support this agreement in varying load conditions of the network and multiple data connections on the same node. This needs a negotiation algorithm between the end-system service provider and the network regarding the aggregate characteristics of the traffic, a scheduling algorithm inside the provider and a feedback algorithm from the network about the network status which helps the QoS management algorithm to decide the further course of action (feedback portion of the survey is presented in the next section). Few people worked in this area of resource allocation and scheduling depending on the QoS contract with the user and vary the QoS requirements depending on the network load conditions. Most of this work did not get much attention because of its hard scheduling algorithms, which cannot be met in real-life scenarios [13], and the others were developed on a custom specific protocol suites [14].

ST-II: [15] ST has been developed to support efficient delivery of streams of packets to either single or multiple destinations in applications requiring guaranteed data rates and controlled delay characteristics. ST is an Internet protocol at the same layer as IP. ST differs from IP in that IP, as originally envisioned, did not require routers or intermediate systems to maintain state information describing the streams of packets flowing through them. ST incorporates the concept of streams across an internet. Every intervening ST entity maintains state information for each stream that passes through it. The stream state includes forwarding information, such as multicast support for efficiency and resource information, which allows network or link bandwidth, and queues to be assigned to a specific stream.

Transport protocols above ST include the Packet Video Protocol and the Network Voice Protocol, which are end-to-end protocols used directly by applications. ST provides applications with an end-to-end flow-oriented service across an internet. This service is implemented using objects called 'streams'. ST data packets are not considered to be totally independent as are IP data packets. They are transmitted only as part of a point-to-point or point-to-multi-point stream. ST creates a stream during a setup phase before data is transmitted.

Problems with ST-II:

- Once the bandwidth for a stream has been agreed upon, it is not sufficient to rely on the originator transmit traffic at that rate.
- The interface between the agent and the network is very limited (our feedback mechanism is useful here).
- The simplex tree model of a stream does not easily allow for using multiple paths to support a greater bandwidth (group management is helpful).

- In case a stream cannot be completed, ST does not report to the application the nature of the trouble in any great detail (error reporting is simple with negotiation protocol).

RFC1046 [16] was intended to explore how Type-of-Service (TOS) might be implemented in the Internet. The proposal describes a method of queuing which can provide the different classes of service. The Type-of-Service (TOS) field in IP headers allows one to chose from none to all of the following service types: low delay, high throughput, and high reliability. It also has a portion allowing a priority selection from 0-7. Priority service should allow data that has a higher priority to be queued ahead of other lower priority data. It is important to limit the amount of priority data. The amount of preemption a lower priority datagram suffers must also be limited.

RFC1349 [17] changes and clarifies some aspects of the semantics of the Type of Service octet in the Internet Protocol (IP) header. The handling of IP Type of Service by both hosts and routers is specified in some detail. This memo defines a new TOS value for requesting that the network minimize the monetary cost of transmitting a datagram. A number of additional new TOS values are reserved for future experimentation and standardization. The ability to request that transmission be optimized along multiple axes (previously accomplished by setting multiple TOS bits simultaneously) is removed. Thus, for example, a single datagram can no longer request that the network simultaneously minimize delay and maximize throughput. This memo again divides the TOS octet of the IP into three fields to represent precedence (bits 0-2), TOS (bits 3-6) and future expansion (7th bit). Even this RFC suffers the same disadvantages of RFC1046.

Resource allocation algorithms

Bandwidth allocation schemes [18] operating at the connection level allocate (reserve) network resources when the connection is being established on the basis of the bit rates requested by the source. Peak bit rate reservation, the simplest one can think of, is relatively inefficient for burst-type data traffic. Over booking (i.e. bandwidth sharing between variable bit rate connections) can be used for source types which are able to specify the variability of the bit rate at connection setup. The bit rate variability of a source can be specified using peak cell rate, sustainable cell rate, and burst tolerance [19]. The sustainable cell rate provides an upper bound on the conforming average rate of a connection; the burst tolerance limits the time a source is allowed to send at its peak cell rate. The connection admission control (CAC) algorithm decides on accepting an incoming connection on the basis of the peak cell rates and the sustainable cell rates of all existing and the new connection.

Bandwidth allocation at connection level for a link with small buffer: Statistical multiplexing with small buffers can function with satisfactory efficiency for connections with low-peak bit rates and small variation in the bit rate. However, the link utilization decreases strongly for data traffic with high levels of burstiness and high-peak bit rates. Significant increases in the utilization can be obtained if the fact can be exploited that many data applications function satisfactorily with a low QoS (i.e. with high cell loss probabilities).

Peak bit rate reservation at the burst level (the Fast Reservation Protocol): The Fast Reservation Protocol [20] is a procedure that allows users to negotiate temporary changes to the bit rate with the networking during a connection. Before

a burst is transmitted, a resource management cell, which contains a bit rate request on the affected connection along the path taken by the connection, is sent. After receiving this "reservation cell," an attempt is made in every node to reserve the bit rate required for the transmission of the subsequent burst. If the bit rate cannot be reserved on some link, a negative response is returned immediately and the source can retry later. If all nodes are able to provide the requested bit rate, a positive acknowledgment is sent by the target node to the source, upon which the source can then begin its transmission. Many of the data applications are relatively tolerant against delays. The fast reservation protocols avoids traffic separation (i.e. the reservation to carry only traffic belonging to the same QoS class over a particular link) between bursty data applications of this type and applications with real time requirements. The efficiency of this method deteriorates drastically for large round-trip delays of the reservation and acknowledgment cells.

Resource allocation at connection level for a link with large buffers: The resource "link bit rate" can be utilized optimally only when the node is equipped with large buffers that can store excess data in large queues during periods of burst level congestion. With large buffers, the effect of burst-level congestion is no longer catastrophic cell loss but an added delay due to the buffering of the excess traffic. With the implementation of the large buffers, we can trade off increased cell delay against improved link utilization. For systems with large buffers, additional parameters which influence burst level fluctuations (e.g. mean burst length, distribution of burst length, auto correlation) have a significant impact on delays and cell loss and can no longer be neglected. Another problem with large buffers concerns cell delay variation: the traffic profile of a connection may be significantly altered after passing through a large multiplex buffer so that the connection's traffic parameters

(e.g. peak cell rate or maximum burst size) are no longer valid.

Window or buffer allocation: Another approach to control cell loss in systems with large buffers has been investigated in [21]. This approach uses the fact that most transport protocols have some type of window mechanisms (i.e. only a limited amount of data can be present in the network at any given time). In the worst case, all this data is contained in a single buffer. This gives rise to a simple rule for dimensioning: allocate as much buffer memory to a connection as determined by the window for the application or allow each connection only a window size in keeping with available memory. If, however, an attempt is made to allocate the buffer only at the connection level, it can happen that memory sizes are required that cannot be technically realized, and that would, in any case, cause unacceptably large delays.

Closed loop reactive load controls (or feedback control algorithms)

Two types of closed-loop reactive load controls are currently under discussion for ATM networks in the ATM Forum: link-by-link, per connection credit-based flow control mechanisms and end-to-end rate-based load control mechanisms. Credit-based mechanisms are similar to flow control mechanism in existing data networks. With rate-based mechanisms the maximum rate at which a source may send can be dynamically adapted depending on network load conditions. ITU in recommendation I.371 provided the possibility to support a rate-based reactive load control mechanism by means of forward explicit congestion notification (FECN). Resource management cells indicating congestion could also be sent directly from congested nodes to the sending terminal, thus providing backward explicit congestion notification (BECN). Refer to [22] for the comparison between rate-based and credit-based

schemes and in turn between FECN and BECN.

FECN[18, 23] is an end-to-end scheme in which most of the control complexity resides in the end systems. When a path through a switch becomes congested, the switch marks a bit in the header of all cells on that path in the forward direction to indicate congestion. The destination end systems monitor the congestion status of each active virtual connection and sends congestion notification cells in the reserved direction on each active virtual connection to inform about the congestion status.

In BECN congestion information is returned directly from the point of congestion back to the source for each virtual channel. The source adjusts its cell transmission rate on each virtual connection in a similar manner to FECN. BECN requires more hardware in the switch to detect and filter the congestion state, and to insert cells indicating congestion into the return path, but it is capable of reacting to congestion faster than FECN. Also, since network itself generates the congestion feedback information, it is more robust against end systems that do not comply with the requirements of the scheme.

The Credit-based approach [24] is a link-by-link window flow control scheme. Each link in the network runs the flow control mechanism. A certain number of cell buffers are reserved for each virtual connection at the receiving end of each link. One round-trip's worth of cell buffers must be reserved for each connection, so the amount of buffering required per connection depends upon the propagation delay of the link and the maximum required transmission rate of the virtual connection.

The group resource management algorithm

One can perceive any application starting from a file-transfer protocol to a video collaborative application as a collaborative application sharing resources. By managing the resources in a collaborative manner in the network and on the host, the resources can be used efficiently. We would like to demonstrate this idea with the help of a video collaborative application.

4.2 Summary of the related work

In this section we summarize the QoS architectural work. Research groups have taken different approaches to address the fundamental questions in QoS architectures. Two such approaches are from IETF and ATM forums. These approaches are service-class-specific (a service class is a group of applications with like-behavior). It is difficult to represent the orthogonal and competing requirements of an application into a service-class: sometimes an application might fall into several incomplete different service-classes leaving the ambiguity of choosing the closest service-class to the application user. Also, it is difficult to accommodate the dynamic nature of the applications, such as the behavior of WWW application, in a static service class. In both of these approaches, it is not possible to use feedback from the current status of the network and the hosts to dynamically modify the application requirements. Even if where a weak feedback mechanism is available, the integration is solution-specific. Hence, they are not inter-operable with other solutions. Existing solutions do not provide a clear mechanism to characterize the requirements of a distributed application. For example, in existing approaches it is difficult to specify the synchronization between two different connections in a distributed appli-

cation. Inter-operability, handling of different networks, and scalability to multiple, different, and concurrent applications is considered an unsolved problem in existing solutions to QOS guarantees.

The ATM Forum proposes a flat classification of applications† depending on the type of traffic the applications generate, timing recovery requirements in the applications, and the type of connections required (connection-oriented or connection less). The ambiguity involved in matching a service class to an application [25] shows that this classification is not complete. One such example is the transfer of an MPEG-2 stream over ATM [26], where the application has multiple choices of AALs to use, but no single one can provide a solution to all the requirements involved in supporting this application.

The IETF also uses flat classification scheme for the services it can support [27]. Two examples where this model fails are IRI and a multiple-priority-connection application (as mentioned in [25]). In IRI two different streams exist for audio and video. These two fall into different classes namely guaranteed (for audio) and predictive (for video). These two streams need to be synchronized at the receiving end. In the current IETF model, it is not possible to specify and provide such a request. The second example is related to mixing connections with different priorities into the same class. It is again not possible to specify such requirements in the application-flow [28] specification.

Other groups such as Tenet [29], TIP [30] proposed a QoS architecture for carrying real-time traffic. Their protocol suite does not provide a solution to the classification of other applications or provide a clear definition of the merge of these other applications into their architecture. We assume that they intend to use differ-

†The existing QoS architectures do not distinguish between an application and a connection. In our architecture we make a clear distinction between these two.

ent protocol architectures for different classes of applications. The method of using different protocol suites for different classes of applications aggravates the problem of maintaining a relation between connections, which is common in collaborative environments.

All the above QoS architectures have their own QoS communication protocol suites, which are meant to support the 1:1, 1:M and M:N application communication between the source(s) and the destination(s). Many QoS communication protocols are proposed for 1:1 communication, such as Tenet's RCAP [31], ATM's Q.2931 [32]. The main disadvantage of these protocols is that they are not scalable to the other two-communication architectures. The scope of some of these protocols are limited by the architectural support from their data communication counterparts (for example, Q.2931 over ATM). ST-II, which was developed to manage the resources [15] in the early ages of Internet (before consolidating the idea of multicasting), makes a point-to-point, source invoked, reservation between the 1:M communicating parties. The disadvantages of ST-II are that it assumes that the source has complete knowledge of its M destinations, which translates into interrupting the source whenever a new connection is added or deleted, and the reservation available to all the destinations is the minimum of all the supported connections. The above two problems are rectified in RSVP by invoking the reservations from the receiver side and by maintaining a soft state of a connection, and by using filters to scale the source emitted stream of data depending on the receiver capabilities. Because of such scalable concepts introduced in RSVP, it can support M:N communication architectures. Though RSVP provides a good QoS communication support, it does not address the issues of QoS inter-networking, group reservation, dynamic reservation for the applications, and the end-user to end-user QoS guarantee.

5 Outline of the thesis

In the next chapter, we present the preliminary experiments conducted which identifies the missing QoS components in the existing high speed networks. In chapter III, we study the issues involved in developing a high quality network and propose our solutions to these issues. In chapter IV, we discuss the design methodology for the QoS architecture presented in the previous chapter. In chapter V, we present the experiments conducted on this preliminary implementation of QUANTA. In the last chapter, we present the conclusions, the limitations of QUANTA and the future enhancements that can be added to QUANTA-like architectures.

CHAPTER II

EXPERIMENTAL BASIS FOR ARCHITECTURAL REQUIREMENTS

Beginning: A journey of thousand miles must begin with a single step.

— Lao-Tzu

An application's performance can degrade mainly for three reasons: end-system protocol behavior in high speed networks, the host load condition, and the network load condition. In this chapter, we investigate the effects of these factors on an end-to-end application. From the observations made from the experiments conducted in this chapter we define the missing components in the current protocol architectures to provide tighter control on the QoS guarantees. The proposed QoS architecture is independent of the testbed machine architectures, excepting the values of the bounds imposed on the QoS by the end-system machine architecture's limitations.

In the first experiment, we run a test application between two SUNSparc 10s under no CPU and network load condition (no-load condition); these results are used to identify the behavior of the end-system protocols for different control

parameters and host-machine limitations. In the second set of experiments, we load the receiver side CPU to test its effect on the application behavior. In the third set of experiments, we observe the effect of network-load on the end-system application behavior. We show how the resultant degradation ends with Quality of Service (QoS) perturbations in the application. From the results obtained in the above experiments, we identify the components in the current protocol-suites missing to obtain end-system QoS guarantees. Some of the experiments we present in this chapter are conducted on both Synoptics LattisCell 10104 and Fore Systems ASX 100 switches. The observations using either switch conform to the results we present in this chapter, excepting for the maximum throughput observed at the application-level due to the differences in the design of the end-system ATM cards. We adopt to Fore Systems solution because of the API they provide to develop our own application code.

As the representative High Speed network, we use an ATM LAN with TCP/IP as the end-system protocol-suite. For the no-load condition experiments, we use TCP(UDP)/IP over AAL5/ATM and direct AAL5/ATM. We use UDP/IP over AAL5/ATM only for the first set of experiments, and for the remaining experiments, use only AAL5/ATM directly, as it is equivalent to the above but with lesser protocol overhead. We experiment with TCP/IP/AAL5/ATM (referred to as TCP), and AAL5/ATM (referred to as direct) protocols under no-load, various CPU, and network load conditions.

The organization of the chapter is as follows: in section 2 we present the testbed used for this work and discuss the relevant background to understand the protocol behavior in different experiments. We discuss in section 3 the translator which will transform the requirements of an $M : N$ application to network QoS

parameters. Section 4 relates the effects of protocol behavior, host CPU load, and network load conditions on the application QoS. We use these results to deduce the modifications needed in the new generation of end-system protocols. We recapitulate the necessity of the proposed QoS components with a case-study in section 5. A summary of the QoS components is in section 6. Conclusions are presented in the last section.

1 Testbed and protocols

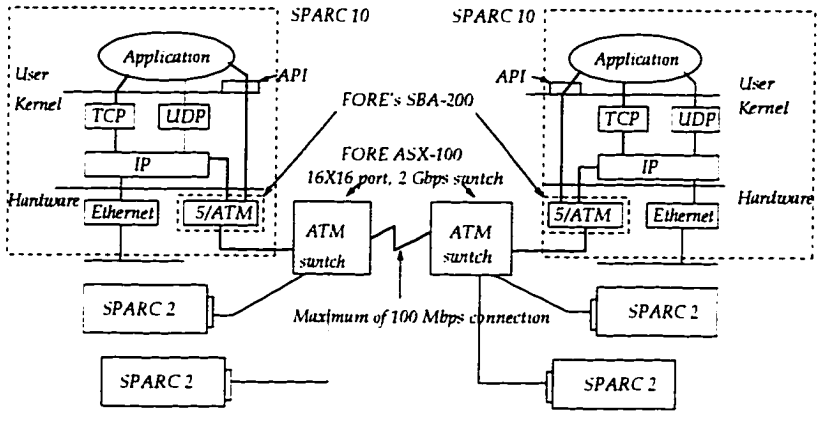


Figure II.1: A detailed testbed used in this work

Figure II.1 presents the testbed used in this work. We have two 16 × 16 port Fore Systems ASX-100 switches connected in tandem. A SUNSparc 10, two SUNSparc 2 workstations are connected to each of the switches. These workstations use Fore Systems SBA-200 ATM cards. The maximum bandwidth between the workstation and the switch and between the switches is 100 Mb/s. We ran the application both through TCP(UDP)/IP running over AAL5/ATM and through AAL5/ATM [33, 34]. When the application runs over TCP(UDP), it uses an end-to-end ABR

(Available Bit Rate) [35] connection, whereas when it uses direct AAL5, it has the option of selecting either a CBR (Constant Bit Rate) or an ABR connection.

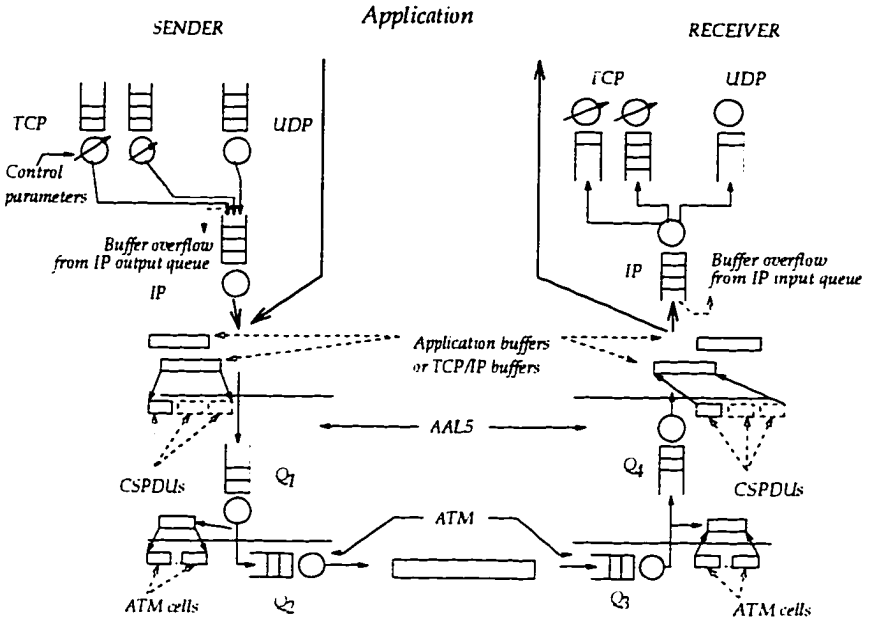


Figure II.2: End-to-end flow of data on an ATM network

Figure II.2 shows the interaction between different protocols and their individual behavior. Every protocol has some control parameters which can be used to adjust the dynamic behavior of the application. In TCP these control parameters include end-to-end variables such as send, receive and congestion windows, retransmission timers, etc., and they also contain algorithms such as slow start and Nagle's algorithm. These control parameters are maintained on a per-connection basis. They can be used to make an end-to-end connection react to congestion in the host and the network environments. Unlike TCP, UDP does not have elaborate control parameters to make its connections lossless. Hence, it depends on the underlying protocols to behave as lossless as possible. Operating System (OS) related parameters such as STREAMS parameters effect the protocol behavior and

hence the dynamics of the end-user application Quality of Service. In a STREAMS protocol implementation, such as in Solaris OS, the protocol modules are linked to each other by a set of incoming and outgoing queues. These queues have lower (Low Water Mark [LWM]) and higher (High Water Mark [HWM]) limits on the amount of data they can store at any time. Once HWM of data are present at the interface queue, STREAMS back pressure algorithms prevents data from entering the queue until the accepting module drains data in the queue to reach LWM. Refer to [36] for more details.

Both TCP and UDP send their data to IP where no distinction is made between the connections from the two protocols. As a result, IP might generate varying delay to different connections and provides unpredictable losses. IP does not inform the user of loss of data. Therefore, these losses in case of UDP result in loss of application packets. In TCP [4, 10] losses are observed at the end of one or more round-trip times (RTTs), which will result in retransmission and hence the reduction in throughput. The congestion control algorithms in TCP may not be as helpful as the congestion avoidance algorithms such as slow start [12]. This is because of the high latency $\times$ bandwidth on ATM networks. The ATM community is looking at rate-based [37] and credit-based [38] congestion avoidance algorithms [39]. Commingling a high data rate UDP application and a TCP application, as we will observe in the following sections, spoils the QoS of both the connections on the sending end-system because of the erratic loss of data in IP.

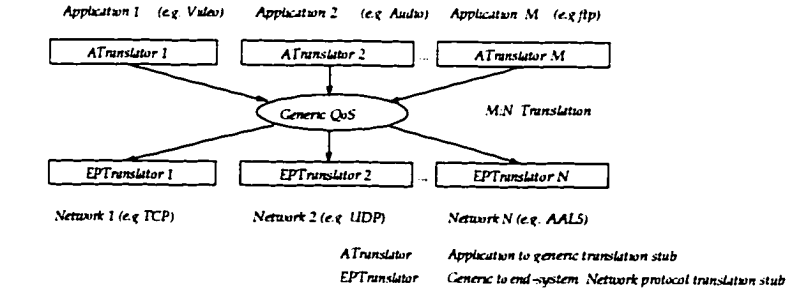
In our environment IP feeds these data to the ATM Adaptation Layer 5 (AAL5). A representation of the data flow through the end-to-end AAL5s is given in Figure II.2. A user data block submitted to AAL5 (either through TCP/IP, or directly to AAL5) is segmented into 9148-byte (most of the time) AAL5 CSPDUs [33]

before queuing in Q_1 as shown in the figure. If the user data is not rate controlled, the Q_1 might overflow resulting in the loss of data on the sender side itself. These CSPDUs are segmented into 48-byte cells and are queued in the ATM layer (Q_2) to be sent across the network. If no per ATM connection queues exist, Q_2 might overflow if the combined data rates of the applications are high compared to the number of buffers allocated to Q_2. Loss of cells might occur on the network if the application exceeds the requested bandwidth or if the network is congested. On the receive side, the cells are reassembled into a CSPDU by ATM and given to AAL5. AAL5 then assembles these CSPDUs to form a user packet. CSPDUs are dropped in ATM (at Q_3) if one or more cells belonging to a CSPDU is lost: this loss of CSPDUs results in dropping the user packet (at Q_4).

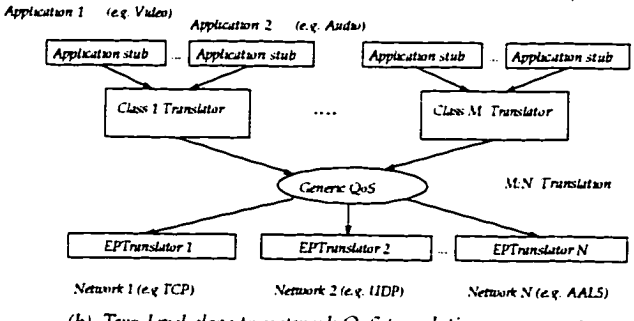
Loss of data anywhere in the queues will reflect as an QoS degradation to the end-system application user. With so many queues and dependencies, the newer generation of the protocols needs to provide a tighter control to be able to guarantee application QoS, which is the major consideration in this work.

2 Application's versus protocol's view of QoS

An application defines the region of operation in terms of its user-level QoS parameters. The application expects the service provider (both the sending and receiving end-systems and the network) to provide QoS within this region during its service. The protocol control parameters, the host-system and the network traffic pose limitations on the application performance. As a first step towards the application-oriented QoS architecture, we propose a component which incorporates the knowledge of the protocol control parameters and translates the application QoS into a set of protocol control parameter values. The translator sets the end-to-end



(a) Two -level application to network QoS translation component



(b) Two-level class to network QoS translation component

Figure II.3: Interface translation from the application to the network

protocol control parameters and OS dependent parameters and hides the relation between these parameters from the application user.

If we have M different applications and N different protocol suites, the translation component should contain M×N translation stubs, to achieve the complete translation set. We propose a two-step translator, which translates the application QoS into a set of generic QoS parameters, such as throughput, loss, delay, RTT [ATranslator] and translates these generic QoS parameters into end-system protocol-dependent control parameter values (EPTranslator). This method calls for $M+N$ translation components, as shown in Figure II.3.a. The translation between the application to generic is dependent on the application characteristics. Hence, M increases linearly with the increase in the number of applications. To control M, we classify applications into different classes as shown in Figure II.3.b. We group protocols into connection-oriented protocols (e.g. TCP, TP4), and connectionless protocols (e.g. IP, IPX). But the subtle differences between the protocols (such as TCP is stream-lined protocol, where as TP4 maintains the integrity of the protocol data unit) need to be treated differently as far as it concerns mapping the QoS onto control parameters.

3 Factors influencing QoS

In this section we quantitatively demonstrate the effect of protocol control parameters on the application QoS. In the process we also identify the host machine limitations such as the maximum throughput one can obtain from the testing host machine. We identify the user-submitted block size as a control parameter and present how it plays a major role in the application's QoS. From these observations we identify missing components in the current protocol suite and present them as

part of our new architecture. We then incorporate one of the components (Rate component) in our test application to observe the improvements and to identify other missing components.

The application running in these experiments is thoroughly modified version of the *ttcp* program*. The command-line options to this application are the size of the user data block, the number of user data blocks to be sent, and the rate at which these blocks are sent (in the direct case, the last parameter will be the peak bandwidth reserved for this application). When it is running on AAL5 directly, it can reserve the requested bandwidth in the case of CBR. We set the TCP and UDP control parameters using *ndd* command, which is available on Solaris OS. These protocol control parameters include, in case of direct, the application requested bandwidth (or data rate), and the application submitted data-block size. In the case of TCP, these control parameters include MSS (Maximum Segment Size), the receive and the send window sizes, and the user block size. In this work we fix MSS to 9148 bytes to obtain maximum throughput.

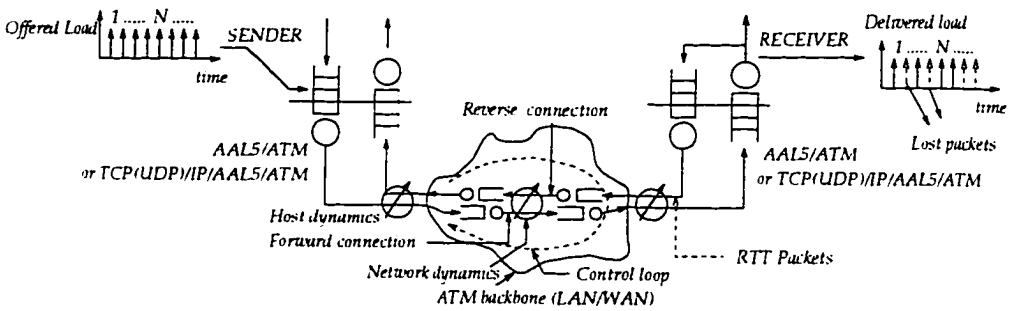


Figure II.4: An insight into the QoS parameter significance on an ATM network

*A Solaris version of this program can be obtained from ftp.cs.odu.edu: /pub/sudheer/modttcp.c.

On the receiver side of this application, we measure throughput, delay, and loss (number of retransmissions in case of TCP or the amount of data lost in case of UDP), and on the sender side, we measure RTT. These parameters capture the dynamic behavior of the host and the network (refer to Figure II.4). Throughput is the amount of data received per second, which is measured on every N^{th} packet (as shown in Figure II.4) and averaged over the total communication time. The throughput requirement of an application reflects the amount of host, and network buffers occupied by the application at any given instant of time. In ATM-like networks, this parameter is also a measure of the amount of bandwidth to be reserved for this application. A low maximum observed throughput on a host machine implies system deficiencies such as a lesser number of system buffers and host-network interface problems. A counterpart to throughput is loss. Loss in our work is defined as the percentage of data lost during the total communication time. We identify data lost on the receive side as shown in Figure II.4 to calculate the loss percentage. Loss, in case of TCP, is observed as more number of retransmissions which shrinks the TCP congestion window and in turn reduces the end-to-end throughput. This parameter in case of UDP or AAL5/ATM connection reflects reduced throughput due to unpredictable loss of data. Delay is the average delay incurred for N packets, which is averaged over the total communication time. Delay parameter is a measure of the queuing delays in the end-system protocol suite, delay at the host-network interface, and delay on the network. This parameter relates to the dynamic behavior of the end-system protocol suite such as the TCP-retransmission algorithm. RTT is the amount of time taken for the N^{th} packet to reach the receiver and back to the sender; the average RTT is calculated over the total communication period. RTT gives the size of the control loop of the application as shown in Figure II.4. In ATM-

like networks, RTT tells the network element what sort of trade-offs it can make between responsiveness, buffer size requirements, available bandwidth, and number of connections it can support [35].

In our experimental set-up we use uniform loading because our intention is not to provide an analysis of a particular application, but to provide a base-line. We measured throughput as the average over a set of every 50 packets and observed the delay for the 50^{th} packet. We ran this experiment for 200,000 packets to obtain enough knowledge about the statistical behavior of the protocol suites at high data rates and determine the average values. In the case of UDP, we calculated loss of data after every 50^{th} packet and present loss percentages over the lifetime of the connection. Every 50^{th} packet is sent back to the sender to measure RTT.

3.1 No-load condition experiments

We divide the no-load experiments into two sets. In the first set, we observe the interaction between the user and the end-system protocol suite. We run the test application on TCP/IP and on UDP/IP protocol suites. We present only the throughput and the loss (in case of UDP/IP) graphs, because the delay and RTT graphs are not statistically significant due to the unpredictable behavior of these protocols (as demonstrated below). In the second set of experiments, we measure the interaction between the application and the network. This is achieved by running the test application on AAL5/ATM. In the process, we also compare these results with that of TCP/IP running on top of AAL5/ATM.

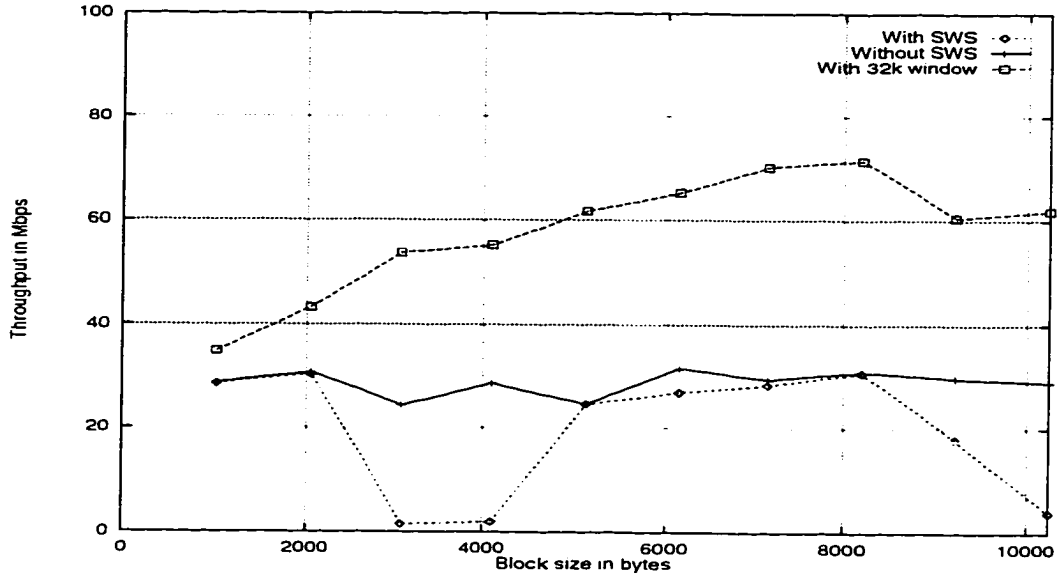


Figure II.5: Effect of TCP control parameters on end-to-end application throughput

Throughput

Figure II.5 gives the average throughput observed on the receive side by varying block size. We recorded a maximum throughput of 30 Mb/s at block size of 8192 bytes. Note the dip in throughput for the block sizes of 3072 and 4096. A phenomenon called "silly window syndrome" (SWS)[40] is the reason for this dip. It should be noted that by eliminating the SWS and increasing the TCP window size (from 8 Kbytes to 32 Kbytes) we can obtain an almost twofold improvement in throughput. As the block size is increased, the processing time per block increases, and hence the throughput decreases. These graphs show that protocol control parameters such as window size, user-submitted block size and algorithms such as SWS will control the behavior of the end-system application. Hence, this experiment suggests that one of the tasks of the generic to network translation component is to set these control parameters to appropriate values.

Throughput observed with UDP is high because of the smaller processing overhead of the protocol. The UDP experiment is also a good test for identifying buffer limitations and other overheads caused by the system. Maximum throughput observed on the receiving side is 70 Mb/s (Figure II.6), but on the sending side, it is almost 120 Mb/s. This difference is due to heavy loss of data in the transit when there are uneven surges in data transfer rates on the sending side (analysis of these losses are presented below). Figure II.6 shows that by incorporating a rate control algorithm that submits data at regular intervals and by increasing the HWM and LWM at UDP-IP interface, data loss in UDP is drastically reduced while there is no change in the obtained throughput.

A steady increase in UDP throughput is shown in Figure II.6 until a block size of 8192 bytes. This is due to the no-processing nature of UDP. UDP submits

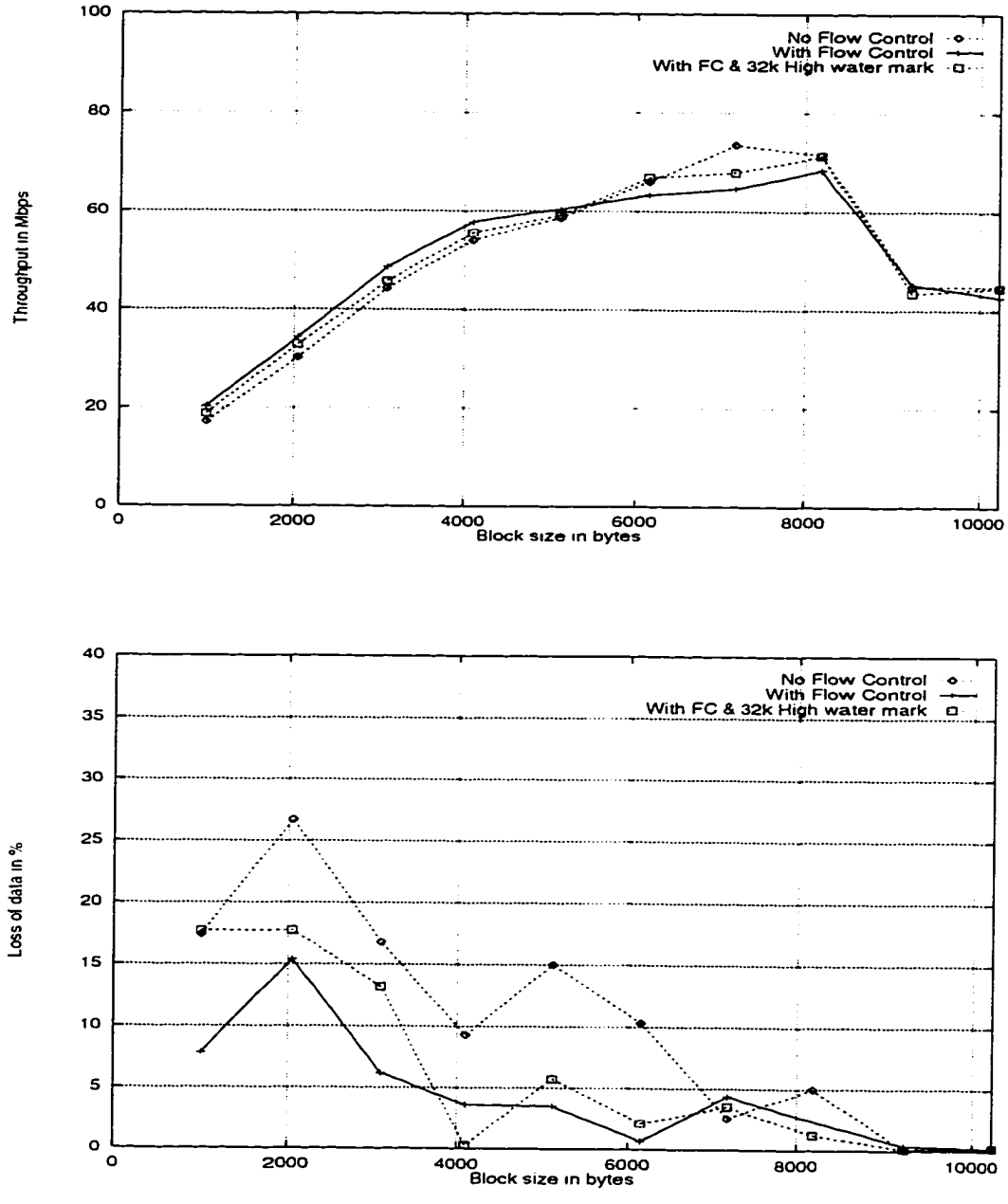


Figure II.6: UDP throughput and loss graphs with and without flow control, and by increasing HWM

packets as they are to IP, which in turn segments them as necessary. On the receiving side these segments are reassembled by IP and are submitted to UDP. As the block size increases, the buffers in the system deplete very quickly and UDP blocks until a fresh quota of buffers is available [36]. Hence, there are higher delays at the sender-network interface.

Loss

Loss of data at high data rates using UDP is due to resource demands on the participating systems and host-network interface rather than any network limitation. Figure II.6 presents the loss percentage versus block size. Loss rates as high as 27% are observed.

Since, loss occurs inside IP, it is left unnoticed by UDP, and hence UDP cannot inform the user about the reason for the loss of data. The loss in IP is due to the overflow of the output data queue, as shown in Figure II.2. This is seen by calculating the number of IP packets dropped during this connection (we used Solaris *ndd* command to get these details.). The loss of data in the IP queue allows UDP to accept data at a faster rate from the user. Such a combination of queue up and dropping of packets in UDP-IP produces erratic time-domain delay patterns (such as in Figure II.7) on the receive side. We can observe from Figure II.7 that the delay builds up and suddenly drops (the effect of HWM and LWM) on the receiving side as well.

To investigate the impact that aspect of UDP-IP interaction has on QoS, we incorporated a rate control algorithm inside the UDP user program in order to send data at a fixed rate depending on the type of the underlying network:

$$Delay = \frac{Block\ size\ in\ bytes}{Allotted\ Bandwidth\ in\ Mb/s}$$

Run forever:

SendData(data)

SendData(data)

SubmitDataToUDP(data)

wait(Delay - Timetaken to submit data -

Other Processing time in the runforever loop)

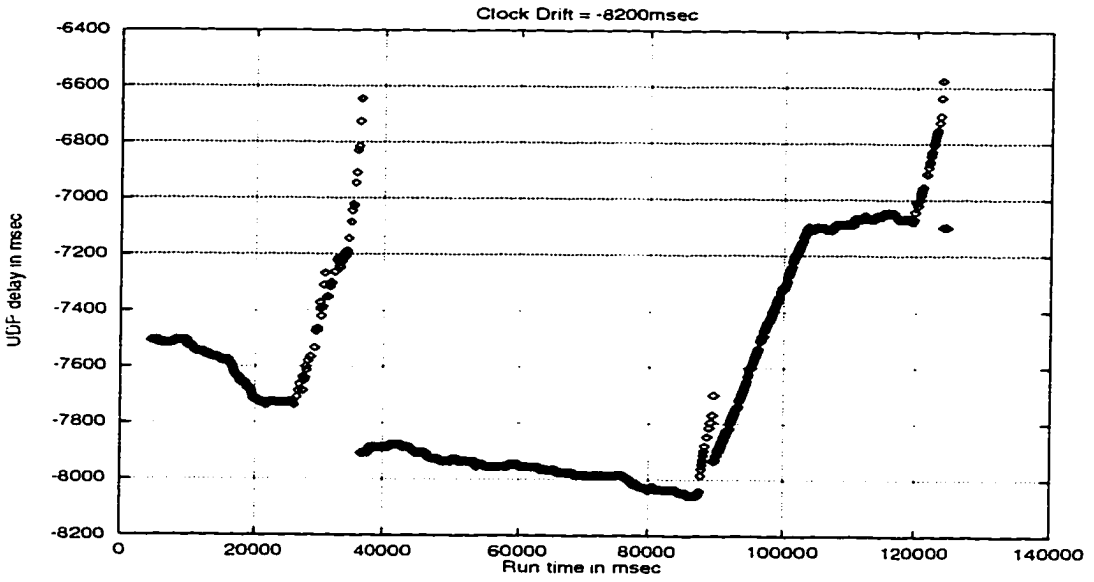


Figure II.7: UDP 5 Kbytes block size time domain delay measurement

This simple algorithm reduced the loss percentage to almost 15% from 27%. The method clearly is not user transparent and is not very accurate. However, were this algorithm incorporated inside the UDP protocol itself on a per connection basis,

we predict that the loss percentage can be minimized. When other connections are on the same host sharing the output queue of IP, it is difficult to maintain the application QoS even with the rate-control.

From the data in the throughput and loss experiments, we conclude that a proper QoS architecture needs to include a resource manager and a local feedback component to isolate the connection, as shown in Figure II.8. Although we could set the initial IOP of an application in TCP using the control parameters, it is difficult to maintain it with interfering connections on the host machine. Hence, even for TCP we need to use the combination of the rate-control, the resource manager and the local feedback to provide QoS guarantees.

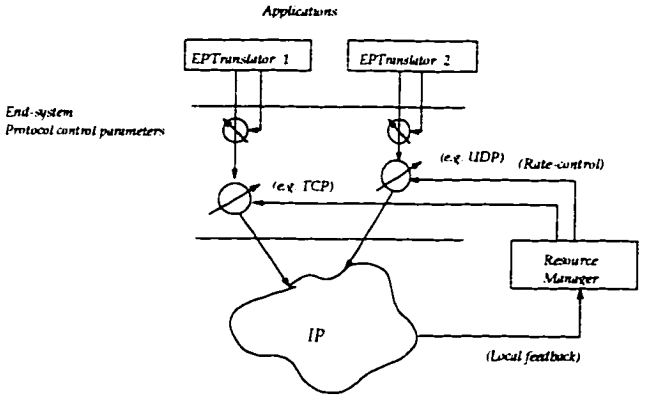


Figure II.8: Proposed modification of the current protocol architecture

No-load case studies with direct and TCP connections

In this section we use different values of target offered load and observe how QoS measures are affected by varying protocol control parameters; in particular, we experiment how different classes of applications behave over direct ATM versus TCP/ATM. The protocol control parameters for the direct case are the application

block size and the requested bandwidth. The bandwidth reserved and requested for an application in direct case is equal to the data rate it attempts to offer (target offered load) using rate control. In case of TCP, the TCP window size is set to 32 Kbytes and MSS is assigned 9148 bytes.

The no-load tests provide a set of baseline results in selecting the control parameters to obtain a specific application level QoS. For both the TCP and direct cases, we measure throughput and delay; for the direct case, we also measure loss, and for selected cases of TCP, we measure retransmissions. In all of these tests, we use a user-level rate control mechanism to control the amount of data submitted per second to match the user-requested bandwidth.

Table II.1 presents the experimental results obtained under no-load conditions for different requested bandwidths and user data block sizes. In this set of experiments, our target is to offer the same amount of load as the requested bandwidth. We present the receiver side throughput, delay, and RTT (on the sender side) for both TCP and direct cases and loss in case of direct connection. The fourth and the fifth columns in the table represent the statistics of the sending and the receiving AAL5. The first portion of the sender side statistics represents the number of CSPDUs given to Q_1 after segmenting (if necessary) the user packet. The second portion of the column represents the actual number of CSPDUs that left Q_1, and the last one represents the percentage of loss in Q_1. Similarly, the first portion in the last column represents the number of CSPDUs assembled by AAL5 before placing the data into Q_4. The second portion represents the actual number of CSPDUs passed onto the user, and the last represents the percentage of CSPDUs lost in Q_4. An intelligent QoS architecture can use this data to select appropriate control parameters. We selected the requested bandwidth to represent different classes of applications.

Common		direct related					TCP related	
Requested bandwidth in Mb/s	Application Block size in Kbytes	ATM delivered throughput in Mb/s	AAL5/ATM sender statistics	AAL5/ATM receiver statistics	Loss in %	Delay on AAL5 in msec	Delay on TCP in msec	TCP delivered throughput in Mb/s
0.5	1	0.47	12800/12109/5.40	12109/12109/0	5.40	**41.6**	127.2	0.49
	8	0.48	1600/1582/1.12	1582/1582/0	1.12	**48.9**	131.0	0.5
	64	0.49	1600/1526/4.62	1526/1526/0	5.00	**128.2**	131.0	0.51
1.5	1	1.40	12800/12125/5.27	12125/12125/0	5.27	15.36	41.52	1.49
	8	1.43	1600/1591/0.56	1591/1591/0	0.56	17.44	43.66	1.49
	64	1.44	1600/1544/3.50	1544/1544/0	3.50	41.34	43.6	1.51
10	1	9.26	12800/11877/7.21	11877/11877/0	7.21	3.04	5.4	9.45
	8	9.33	1600/1561/2.44	1561/1561/0	2.44	2.64	6.5	9.9
	64	9.48	1600/1480/7.50	1480/1480/0	7.50	6.45	6.54	10.09
25	1	**7.32**	12800/12245/4.34	12245/4132/66.26	**67.72**	3.63	4.4	13.47
	8	21.89	1600/1488/7.00	1488/1488/0	**7.00**	1.16	2.82	22.71
	64	21.06	1600/1361/14.93	1361/1361/0	**15.00**	2.26	2.64	24.83
65	1	**5.34**	12800/12800/0	12686/2144/83.10	**83.25**	5.34	3.00	15.50
	8	62.16	1600/1600/0	1600/1600/0	0.00	0.60	1.42	43.39
	64	62.87	1600/1600/0	1600/1600/0	0.00	0.93	1.66	38.75

Table II.1: Comparison of direct and TCP applications under no load condition

For example, 0.5 - 1.5 Mb/s represents MPEG-1 compressed video stream, and 10 - 65 Mb/s represents high data rate applications. Certain data are bold-faced in the table to draw the reader's attention to those values (explained in the next two subsections).

Case (i): 0.5 - 1.5 Mb/s applications

For both TCP/ATM and direct ATM cases, providing the requested throughput for low bandwidth cases is never a problem. Delay increases as user block size increases in the case of direct ATM connection because of the processing overhead. This end-to-end delay is much less in case of 1 Kbyte and 8 Kbyte block sizes (41.6 msec and 48.9 msec) compared to that of 64 Kbyte block size (128.2 msec) because no segmentation and reassembly is done in ALL5. Because of the segmentation and reassembly in AAL5 for 64 Kbyte block size, delay increases by almost threefold. Delay in case of TCP is high compared to that of direct ATM connections, because TCP tries to accumulate MSS worth of data before sending it to AAL5. TCP delay can be reduced by reducing the window size and MSS (if possible). The loss of data in the direct ATM case occurs at Q_1 due to our user-level rate control which may not supply data at the exact requested rate. Loss can be made zero by increasing the requested bandwidth marginally more than the offered load (refer to Table II.2).

Case (ii): 10 - 65 Mb/s applications

For high data-rate applications, the number of interrupts generated on the receive side of the application by the ATM card are high in the case of using a 1 Kbyte block size. This phenomenon leads to dropping packets in Q_4 if the host is not fast enough. Hence, the user-level throughput goes down (5.34 Mb/s for 65 Mb/s case) and losses are high (83.25% for 65 Mb/s case). Delay will also increase because of higher queuing delays at Q_4. We also noticed that using our testbed we

could not observe throughput more than 65 Mb/s, which is the host limitation. By increasing the block size to 8 Kbytes, we are reducing the number of CSPDUs, and hence, the number of interrupts on the receiver side, which will reduce the losses to zero.

In Table II.2 we present the relation between RTT, delay, and losses at different level of target offered load, and requested bandwidth. For 0.5 Mb/s case, high RTT values are observed when the requested bandwidth is the same as the target offered load; this is due to all the resources being allocated for the duplex connection are used by the forward connection (refer to Figure II.4) itself. When the requested bandwidth is more than the target offered load, the RTT becomes decreases less because of the resources available in the reverse direction. Average delays have increased as more bandwidth is reserved, because of the increase in queue sizes allocated for this connection. As observed in case of 8 Kbyte block size, average delays reach saturation for example 130.10 msec: this is because the leaky bucket rate-control algorithm in ATM is becoming affective.

At 65 Mb/s, the target offered load and the requested bandwidth does not effect the delay, and RTT for 1 Kbyte block size, due to losses at Q_4 (buffer overflow). At 8 Kbyte block size, because of the zero data losses, the delay and RTT are comparable. These parameters cannot be improved by increasing the requested bandwidth.

In case of TCP, at low data rates TCP will not encounter losses and hence no retransmissions. We use this information to calculate the percentage of retransmissions in the lossy case of 65 Mb/s with 1 Kbyte block size. We observe 13% retransmissions, which is less compared to 83.25% losses in its counterpart in the direct case. At the same time, TCP adjusts itself to a lower throughput to avoid

direct related					Common	direct related		TCP related	
Target Offered load / Requested bandwidth	Block size in Kbytes	Delivered load in Mb/s (ATM)	RTT in msec in msec	Delay	# Blocks (ATM)	CSPDUs in %	Loss (TCP)	CSPDUs in TCP	Retx. %
0.5/**0.5**	1	0.47	**1060.52**	**39.36**	12800	12109	5.40	**4554**	0.00
0.5/**0.7**	1	0.49	**15.61**	**130.88**			0.00		
0.5/0.5	8	0.50	5672.59	48.9	1600	1600	**1.12**	1600	0.00
0.5/0.7	8	0.49	115.39	**129.10**			**0.00**		
0.5/1.0	8	0.49	69.70	**130.10**			0.00		
65/65	1	5.34	305.72	5.50	12800	12800	**83.25**	**5144**	**13.00**
65/75	1	7.30	270.41	37.52			**77.41**		
65/85	1	7.05	**251.52**	**37.36**			**77.62**		
65/65	8	62.16	4.26	0.99	1600	1600	0.00	1600	0.00
65/75	8	62.59	**4.25**	**1.00**			0.00		

Table II.2: Case analysis direct and TCP applications for different block sizes

losses. This shows the self-healing nature of TCP because of its elaborate flow control mechanism.

From the above observations, we conclude that the application block size and the requested bandwidth play major roles as control parameters in satisfying an application's requested QoS. An application using direct connection should clearly balance the control parameters to obtain the desired QoS. Whereas, an application using TCP need not balance the control parameters at the cost of not obtaining tight bounds on the application QoS. We notice that the maximum data rate a TCP connection supports is around 45 Mb/s, whereas a direct connection supports up to 65 Mb/s on a SUNSparc 10 workstation. Hence, with the given configuration, the combined data rate of all TCP applications should not exceed 45 Mb/s, similarly for direct it should not exceed 65 Mb/s.

These observations lead us to further modify the proposed QoS architecture, as shown in Figure II.9.

The maximum observable QoS parameters on a system are bounded by the kernel-resources, such as buffers and the host machine's architectural limitations. These values will define upper bounds on the QoS which can be requested from that host. Therefore, the resource manager component should consider these host limitations while allocating resources to multiple connections.

The direct connection analysis shows the sensitivity of QoS to network control parameters. For example, we demonstrated this in Table II.2 by showing that zero loss can be obtained by reserving higher bandwidth than requested. Hence, to translate the application QoS into different backbone network protocol QoS parameters, we need an end-system protocol to network translator

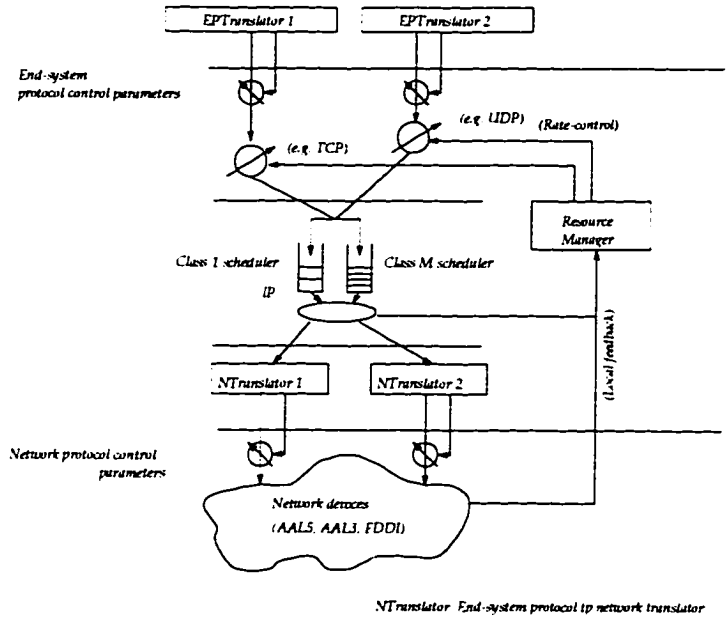


Figure II.9: Further modifications to the protocol architecture

(NTranslator). If the application QoS is sent in generic format, we can avoid two-level translation at the protocol to network interface.

From the loss analysis on the receive side (due to higher number of interrupts), we can conclude that we need to provide feedback from the network device to the resource manager. This information should be communicated to the other-side of the end-system protocol.

As the number of connections through IP increases, the feedback becomes complex and even simple rate-control becomes complicated. This suggests that the control in IP should be performed on a class-basis rather than on a per-connection basis. By selecting the classes which can be recognized at the user-level itself, we can reduce the complexity of mapping between classes at different layers.

3.2 Host behavior experiments

The next set of experiments is to determine the range of impact the end-system load has on the ability to maintain predictable QoS performance. We study the behavior of TCP and direct protocol stacks using host-load condition. Guru Purulkar et al. [41] reported a study on a SunOS 4.1 in which one of their conclusions was: in a UNIX-like Operating System a communication-intensive job receives higher priority over a computation-intensive job because of Unix's dispatcher algorithm. In our host-experiments, we obtained similar results but relate them to QoS performance.

The computation-intensive job is a software decompression program of an MPEG-1 stream, and the communication-intensive job is the modified *ttcp* application. We use 8 Kbyte user data blocks with TCP control parameters set to 32 Kbytes for the window size and 9148 bytes for MSS. For direct connection, we use a user data block size of 8 Kbytes and allocate bandwidth equal to the target offered load. The experiment is run under different receive side CPU-load conditions. From Table II.3, we make the following observations: In the 0.5 - 25 Mb/s range (irrespective of the percentage of the CPU load) the losses are mainly at Q_1 because of the inaccurate user-level rate control at the sender. Delay using TCP is still higher compared to that using a direct ATM connection. The throughput is comparable in both the cases.

For the 65 Mb/s case losses shift from the send side to the receive side because the application is working in the maximum throughput range for that machine, and even a slight CPU load leads to loss of data in Q_4. TCP on the other hand adjusts itself to lower throughput in the process of reducing the number of retransmissions because of the losses on the receive side. AAL5 throughput is still higher than TCP but this might be a bad option in such cases because of random losses. TCP

Common		direct related					TCP related	
Requested load in Mb/s	CPU load in %	AAL5 delivered throughput in Mb/s	AAL5/ATM sender statistics	AAL5/ATM receiver statistics	Loss in %	Delay on AAL5 in msec	Delay on TCP in msec	TCP delivered throughput in Mb/s
0.5	80	0.48	1600/1583/1.06	1583/1583/0	1.06	49.95	130.99	0.50
1.5	80	1.44	1600/1594/0.38	1594/1594/0	0.38	17.75	43.74	1.50
10	80	9.46	1600/1560/2.50	1560/1560/0	2.50	2.67	6.51	9.98
25	20	21.82	1600/1501/**6.19**	1501/1501/0	6.19	1.14	2.78	22.96
	40	21.80	1600/1507/**5.81**	1507/1507/0	5.81	1.20	2.69	23.61
	60	22.03	1600/1494/**6.62**	1494/1494/0	6.63	1.20	2.72	22.94
	80	25.92	1600/1503/**6.06**	1503/1503/0	6.06	1.31	2.63	23.67
65	20	43.50	1600/1600/0	1600/971/**39.31**	39.31	2.94	1.49	37.11
	40	45.77	1600/1600/0	1600/920/**42.50**	42.50	2.87	1.47	36.51
	60	49.98	1600/1600/0	1600/1095/**37.90**	37.19	3.07	1.81	35.85
	80	55.97	1600/1600/0	1600/1228/**23.25**	23.25	1.64	1.40	39.73

Table II.3: Comparison of direct and TCP application at different CPU-loads

has a better delay characteristics because it dynamically adjusts itself to the lossy behavior of the communication path.

The conclusion from these results is that the resource manager needs to reserve resources on the end-system host machines and should accept feedback from the network interface component.

3.3 Network behavior

The final component we analyze to determine what is needed in a QoS architecture to provide predictable performance is the one which relates to network behavior. The following experiment is set to determine the deficiencies in TCP and the interaction of TCP with direct connections under network-load conditions. We present time domain plots of throughput and delay of the end-to-end applications. For a detailed analysis of the network-load experiments, refer to our extended version of this work in [42].

We use ABR connections for the direct case in this experiment: the results of which are shown in Figure II.10. We set up a TCP connection between the two SUNSparc 10 workstations and two direct connections between the SUNSparc 2 workstations. The TCP connection is rate-controlled to produce 45 Mb/s, whereas the direct connections are tuned for 35 Mb/s each. Figure II.10 gives the throughput and delay values for the three connections over a period of time. We intentionally overload the link between the ATM switches to observe behavior of the end-system protocols in adverse conditions and its effect on the application QoS.

As can be observed from the throughput graph in Figure II.10, TCP behaves poorly in combination with the other direct connections even though almost 30 Mb/s bandwidth is available. This is because TCP does not obtain the information on

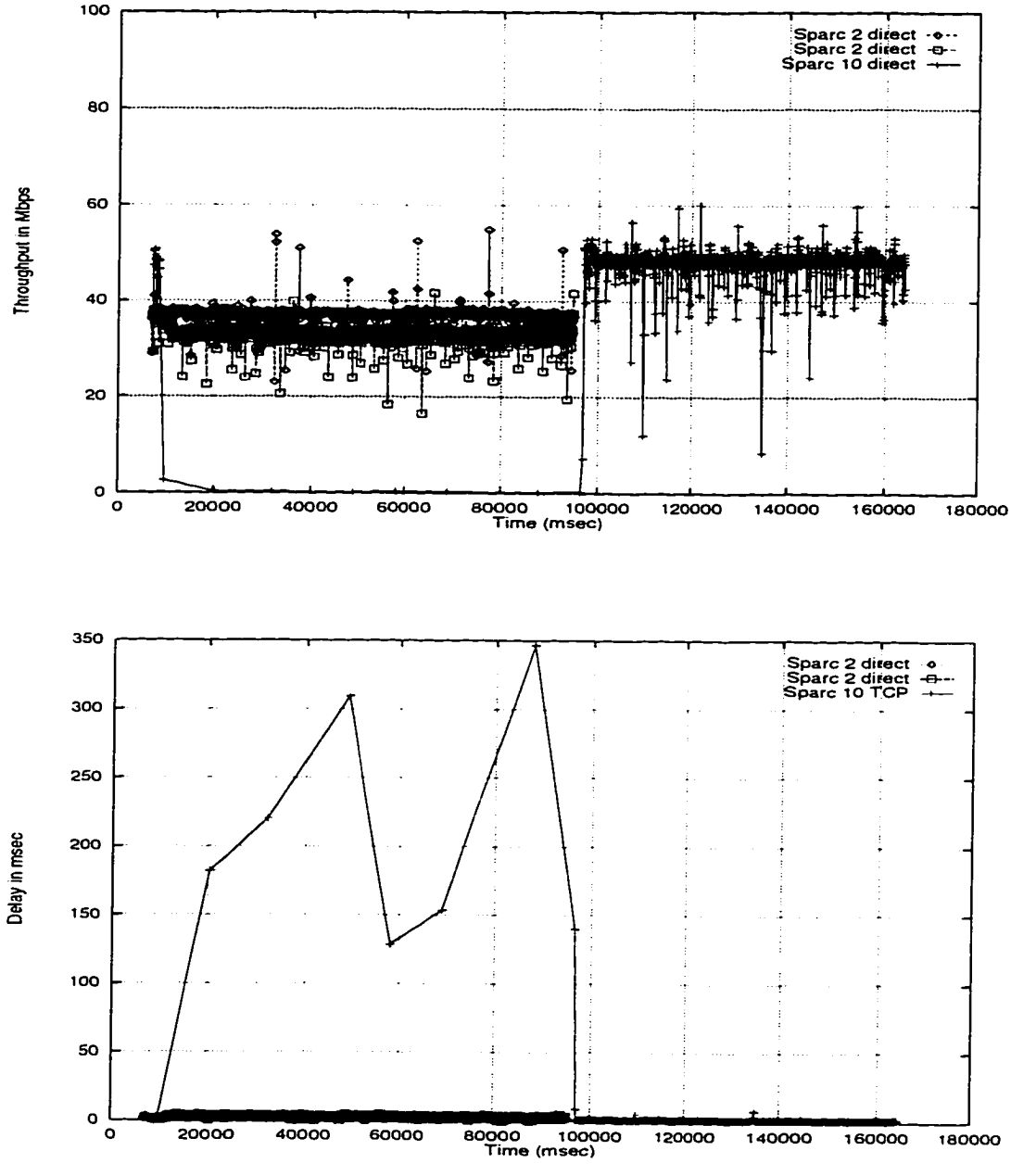


Figure II.10: One TCP at 45 Mb/s, and two direct at 35 Mb/s

the residual bandwidth. As a consequence of low throughput, the average delay for this connection is high until the direct connections are active. As soon as the direct connections are closed, TCP picks up at a high-data rate, and hence lower average delays are also seen. Even at lower data rates for the direct connection, TCP performs the same way. The direct connection on the other hand, works at higher data rates, because it does not have elaborate error-recovery. The penalty paid for the higher data rate in the direct connection is the bursty losses.

This leads us to the conclusion that resource allocation has to be done on a per-TCP connection basis to obtain maximum utilization of the network resources. And the allocation should be adjusted dynamically according to the feedback from the network. These components are incorporated in the complete architecture shown in Figure II.13.

4 Case study

In this section we consider a medical demonstration classroom application[†], where an on-going operation is captured live from the medical theater and is sent to another room for a large group of audience. The application sends large amounts of uncompressed data, expects low latency and small losses. In the following discussion we trace how these requirements can be met assuming we have an architecture such as we proposed in the previous sections. Let us assume that this operation takes 10 minutes and imposes the following requirements on the video quality: the picture should be high quality (24 bits/pixel), frame size should be 320 × 240 with 20-30 Frames/sec (FPS), and the loss should not exceed 15 blocks/sec (each block is 16

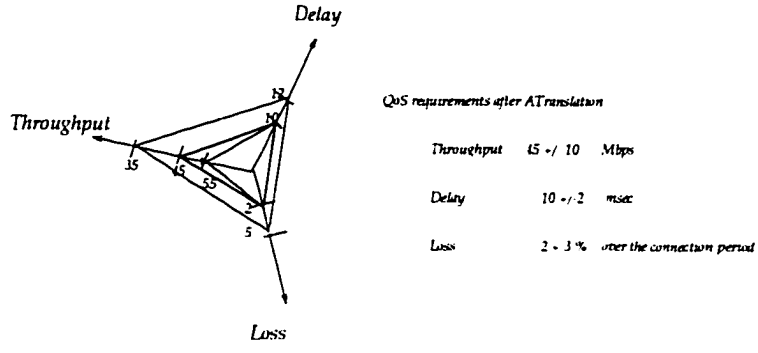
[†]We use this application because our group at ODU is involved in one such project (which is a collaboration between ODU, NASA, and a local medical institute).

pixel × 16 pixel) in the case of 20 FPS and 20 blocks/sec in the case of 30 FPS.

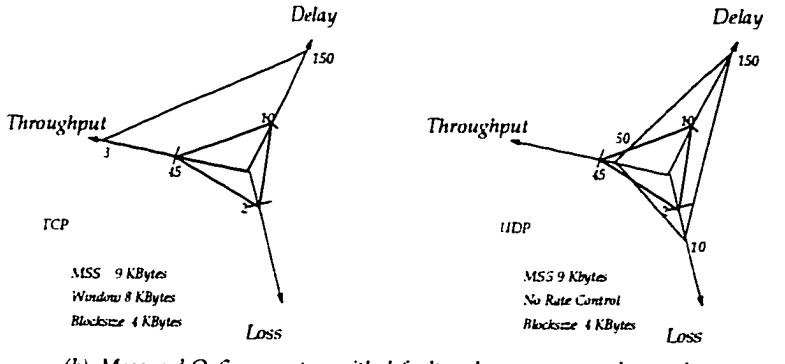
The ATranslator translates the above requirements to the generic QoS requirements as follows: the frame size and the frame quality translates into throughput rates of 45±10Mb/s; the loss encountered should not exceed 2-5% during the connection; and the latency requirements are translated to 10msec±2msec (refer to Figure II.11.a). The ATranslator should be intelligent enough to select the appropriate protocol such as TCP or UDP.

From the selected end-system protocol and from the application QoS requirements, we select appropriate protocol control parameter values using the EPTranslator. In Figure II.11.b, we depict the effect of the default control parameters values on the application QoS in both TCP and UDP cases. As we can see, it is impossible to achieve the requested QoS without the knowledge of the underlying protocol. By properly selecting and setting the control parameters alone, shown in Figure II.11.c, bounds on throughput could be achieved in both TCP and UDP. However, in TCP we can not bound delay characteristics, and in UDP we cannot bound both delay and losses.

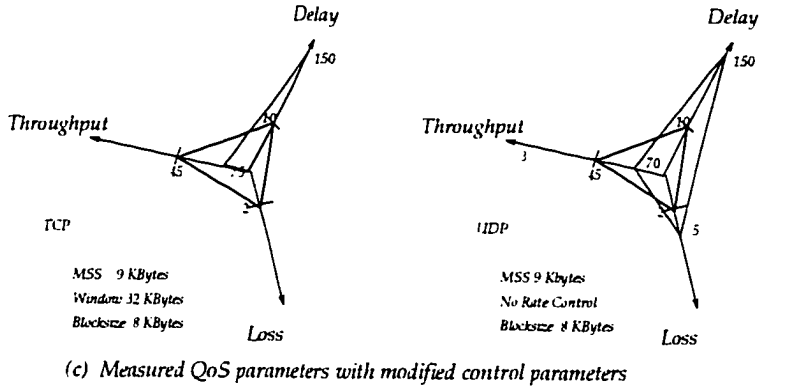
At this point since, we cannot deduce much information on the interaction between the end-system protocol suite and the network protocol suite, we use direct connection, which is equivalent to UDP/IP running over AAL5/ATM with less protocol overhead. The delay will be more by approximately 10msec with the addition of UDP. By providing a simple rate-control at the user-level, we can bound the losses and the delay in direct connection and the delay in TCP as shown in Figure II.12.d. In the previous section, we have shown the relation between control parameters of the end-system protocol suite and the network protocols. These control parameters provide tighter bound on the QoS parameters and also provide tradeoff between the



(a) Medical Demonstration Classroom Application



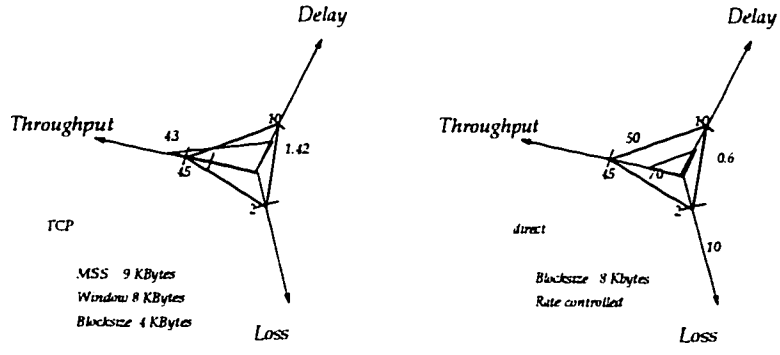
(b) Measured QoS parameters with default end-system protocol control parameters



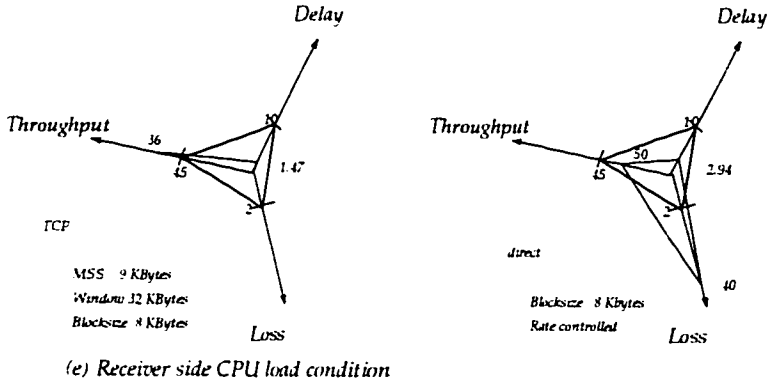
(c) Measured QoS parameters with modified control parameters

Diagrams are NOT drawn to scale

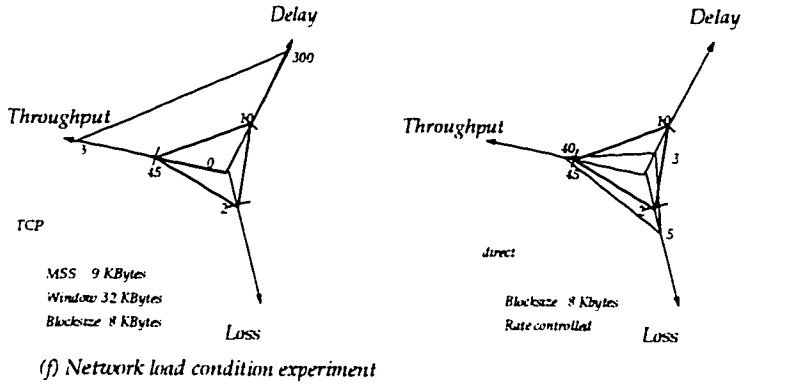
Figure II.11: Example for QoS improvement



(d) Measured QoS parameters with rate-controlling the application



(e) Receiver side CPU load condition



(f) Network load condition experiment

Diagrams are NOT drawn to scale

Figure II.12: Continuation of the example for QoS improvement

QoS parameters such as delay and RTT.

As shown in Figure II.12.e, by incorporating the CPU-load on the receiver side, we can justify the necessity of the feedback component. Though this will cause less impact on the TCP connection, it is very apparent in the direct case in terms of the percentage of total loss of data. The net-load experiments measured the interaction between different end-system protocol suites and the bounds they can provide in a congested network condition. We have seen in Figure II.12.f that TCP provides abysmal QoS performance compared to the behavior of a direct connection, which accommodates itself with a slight increase in loss. This observation consolidates the idea of the feedback component and the resource manager who can translate the feedback into controlling the source stream into the network.

5 Discussion on QoS architecture

In this section, we present a summary of the high-level design of the end-system QoS architecture we propose (refer to Figure II.13). It is based on the conclusions we have drawn from the experiments described in the previous sections. A complete specification and the status of its implementation can be found in [43]. The architecture has three main components: application-level, protocol-level, and global components, as shown in Figure II.13.

Application-level components

We divide applications into different classes depending on their QoS requirements. Each class of the applications talks to its corresponding module in the class-specific interface module in the application-level component. The QoS requirements

are translated into a generic set of QoS requirements by the generic interface. These generic requirements are translated into a set of protocol-related control parameters using data we presented in the previous section and in [44]. These control parameters are sent to the service-providing protocol to set the appropriate values for this connection. The interface will also interact with an application to dynamically adjust its QoS requirements depending on the network, and the host status using feedback from the service-provider.

Protocol-level components

Based mostly on the host, and network behavior experiments, the architecture provides the following modifications to the end-system protocol architecture:

- Rate-control algorithm: Both TCP and UDP require a connection-based rate-control algorithm to limit the user to behave in its requested QoS. In TCP, this algorithm works below the window-based retransmission scheme to retain the flavor of the existing TCP. In UDP, rate-control prevents a user from sending at a higher data rate than the agreed-upon data rate by blocking the application. This scheme reduces the losses in UDP due to uncontrolled transmission of data which leads to buffer overflows.

- Local feedback algorithms: Feedback from the service-provider, such as from IP to TCP(UDP) or from TCP(UDP) to the application user, serves to change the control parameters, and in turn, retains the user requested QoS. For example, IP feedback information could be used in UDP to reduce the data rate of an application to avoid further losses in IP.

- Connection-based monitoring: All the applications with specified QoS require-

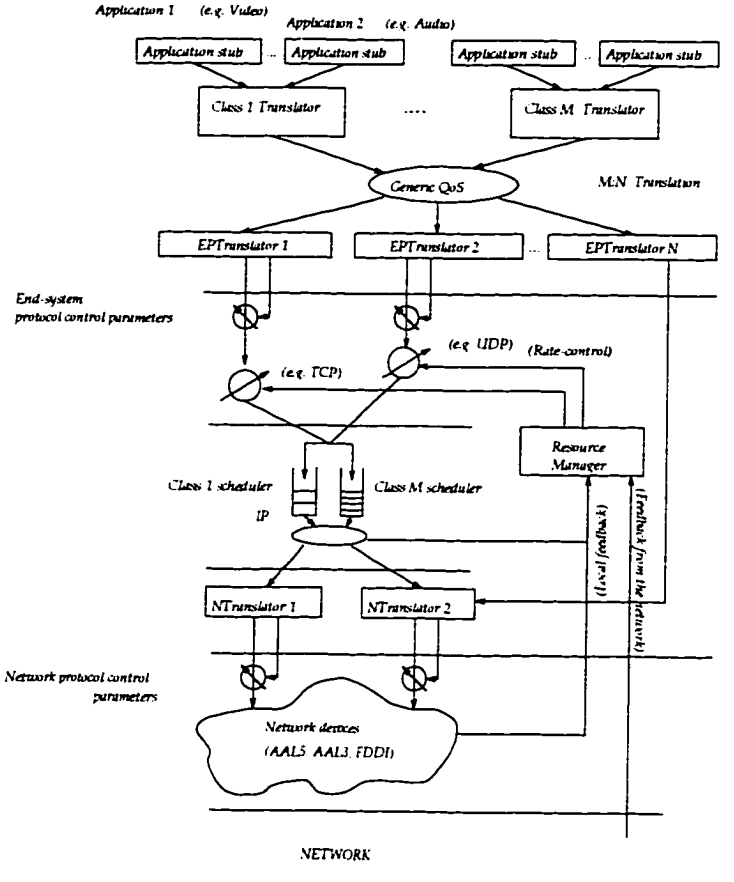


Figure II.13: End-system QoS architecture

ments should conform to the initial negotiation. A deviation from the negotiation will effect the other connections using a common resource pool. Hence, to retain the isolation between the applications, we need to monitor the application to check if it is within its allocated resources. This monitoring is effective if it is done closest to the user application. Hence, we propose to do the monitoring on per-connection basis at TCP or UDP.

- Resource allocation: To avoid the complexity of handling resources on per-connection basis, we provide class-based resource allocation. For example, we have real-time and time-shared classes inside IP. In a real-time class, we schedule the data to meet dead-lines, whereas in a time-shared class, we are interested to obtain a given throughput.

- Class-based monitoring and scheduling: We provide class-based monitoring to avoid overflowing the resources, which affects the QoS of all the applications in that class. The scheduling algorithms we use in each class of applications are different. A scheduling algorithm is dependent on the combined knowledge of all the applications in its class.

Global components

When the status of resources changes, we need to maintain information about all these resources to be able to degrade the performance of an application gracefully. This has to be done on a global basis. These global components include network feedback, resource control, and global monitoring. The network feedback information is used to monitor the status of the network and react to it in order to reduce loss or control delay for an application. Global resource control is used to allocate

resources dynamically to different classes of applications. Finally, global monitoring is used to predict the degradation in the performance of the applications and inform their class schedulers about modifying the scheduling parameters of the algorithms.

This QoS architecture is generic enough to work with a simple link-level protocol like Ethernet protocol, to a complicated protocol such as ATM in a LAN environment. The algorithms we are developing in this architecture are generic enough to be incorporated in any transport-level protocols such as TCP or UDP and network protocols such as IP.

6 Summary

In this chapter we used the application-oriented approach to propose a QoS architecture for a TCP/IP-like end-system protocol-suite. We conducted no-load, host-load, and network-load condition experiments to identify the missing components in the current architecture of TCP/IP. These missing components include a two-level application to network QoS translator, protocol tuning components, local feedback component, and class-based scheduling.

We presented the base-line QoS that can be achieved by an application in a LAN environment. We compared the behavior of an application using TCP(UDP)/IP over AAL5/ATM transport mechanism and direct AAL5/ATM with respect to their control parameters. We identified bottlenecks an unwary user might encounter, such as high delay at higher block size, heavy losses for 1 Kbyte block sizes at high data rates, relation between the requested bandwidth and target offered load. We demonstrated the trade-offs between the QoS parameters with the help of the control parameters, such as obtaining zero loss, reducing the RTT etc. The proposed

QoS system can refer to the tables presented in this chapter to select appropriate protocol control parameters to provide an application specified throughput, loss and bounded delay requirements.

In the next chapter, we use this knowledge in better understanding the issues in developing a QoS architecture. This outline design of QUANTA is expanded in the design chapter.

CHAPTER III

ISSUES, APPROACH AND ARCHITECTURAL DESIGN

Imagination: Raise new questions, explore new possibilities, regard old problems from a new angle.

— Albert Einstein

Glossary of the terms used in this chapter

A service user is one who expects certain services from a service provider. For example, this user-provider pair could be between the application and the TCP/IP protocol-suite or between the TCP/IP protocol suite and the ATM network.

A connection is the path of communication between two end-system entities, which are engaged in some type of communication. A connection will have a forward data path in which it sends data from the transmitter to the receiver and a backward data path in which responses and acknowledgments are sent.

An application is a set of service components (programs or tools) which provide certain services to the application user. A distributed application has one or more nodes involved in communication and will have distributed service components co-operating with each other. This distributed application could be between one-to-one, one-to-many, or many-to-many nodes, also it could contain multiple connections between the service components. In this paper we use the term (unless otherwise specified) application to address a one-to-one distributed application, and a distributed application is a one-to-many or many-to-many distributed application. A connection subtree in a network represents the tree of connections formed through the intermediate nodes with a router (as a root node) and the end-systems (as the leaf nodes) which are participating in a distributed application.

Every connection in an application generates data traffic, which can be characterized by its traffic parameters. Each connection requests certain qualities for its data stream, which can be represented by the QoS parameters. Connections with a similar set of QoS requirements are grouped into service classes.

A QoS architecture defines different QoS components (such as QoS classification mechanism and QoS specification mechanism) to provide support for the QoS guarantees to applications. We use the term architecture synonymously with QoS architecture, unless otherwise mentioned. QoS algorithms or QoS mechanisms implement the services specified by the architecture. A QoS component can be implemented or recognized by many QoS mechanisms. QoS architecture and QoS mechanisms decide the range of different applications they can support. An implementation architecture consists of implementation components which support one or more QoS components. In our discussion, a

component means an implementation component, unless otherwise specified.

We use the term scalability of an architecture or an algorithm to imply their flexibility in accommodating applications with different service requirements.

A connection identifier (CID) is an identifier which uniquely identifies a connection and its parent application in the network and on the host system. A flow identifier (FID) of a connection represents its QoS requirements. A QoS traversal graph (QTG) map of a connection characterizes the connection behavior (in one byte). We use the term QoS identifier (QID) as a combination of FID and QTG.

QoS guarantees are required to provide a fair share of allocation of the network and the host resources to multiple connections. In seeing the same problem from an application's point-of-view, once the host and the network agree to the application's QoS contract, it should be maintained throughout the connection period. QoS could be degraded, for example, due to a misbehaving application which is sharing the resources with the above application or because of the statistical multiplexing of a group of applications. Because of high *latency X bandwidth* in the current technologies, the solutions provided for QoS control should be preventive rather than reactive. Since it is virtually impossible to provide complete preventive solutions in HSNs, the prime goal of new QoS solutions should be reacting quickly to QoS degradation. The two components that dictate the usability of the solution are the architecture and the algorithms used to address these issues. Our goal in this work is to provide one such architecture for the QoS grantee.

As we will observe in this chapter, although there are many studies per-

formed in providing QoS guarantees, none were conducted from the architectural point-of-view (most of them are from the algorithmic point-of-view). Also, the QoS solutions provided are either specific to service classes (such as the ATM approach) or specific to a single class of applications (such as the Tenet, RTP approaches). These approaches do not encapsulate all the applications and their behavior. The issue of scalability of these solutions to a higher number of connections or to a higher number of nodes is also not considered by many of these solutions.

In section 1 we revisit the QoS problem from an application's point-of-view to better understand the issues. In sections 2 and 3 we discuss the issues of isolation of an application and knowledge of application issues respectively in the perspective of the existing solutions, point the disadvantages of these solutions when analyzed from the application's point-of-view, and then provide our solution to address these issues. Section 4 summarizes our major contributions in this chapter.

1 Revisiting the problem

In this section, we present a typical distributed application and provide motivation to the fundamental issues in providing QoS guarantees.

IRI (Interactive Remote Instruction) [1] is a distributed collaborative multimedia classroom, which is designed to support distance learning. IRI can facilitate both teacher/student and student/student interaction through two-way audio and video, and tool sharing. In Figure III.1, we present a teacher's workstation with the emphasis on the communication activity that goes on his end-system.

The out-going connections from the host machine through IP* are the teacher's

*We consider IP because this is the place where we commingle all the connections using end-system transport protocols.

video, the teacher's audio, the collaborative tool generated data, and the global pointer movements. The incoming connections into IP include the student's video, classroom video, the incoming audio, and the data generated from the student controlled tools.

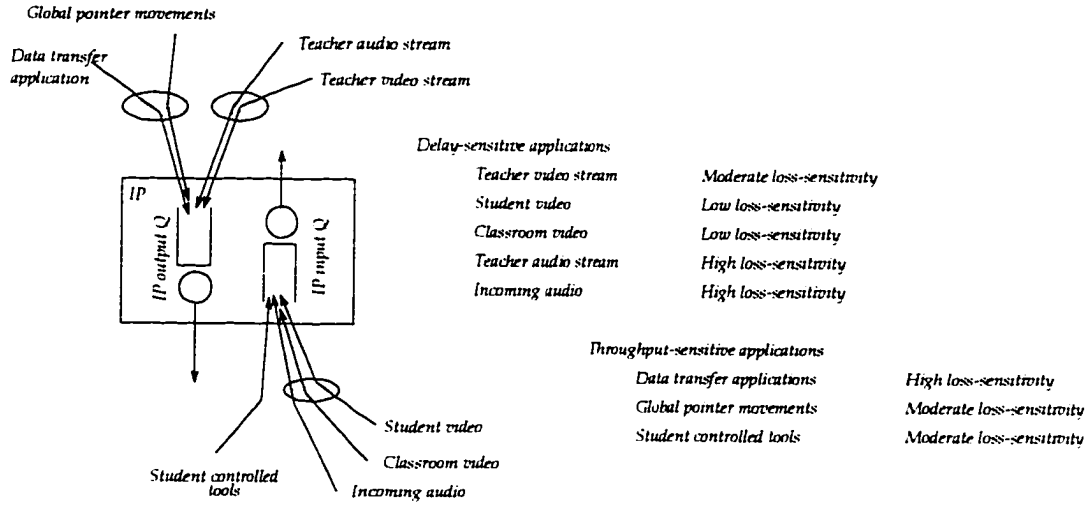


Figure III.1: Different connections going through the teacher's node in IRI

IRI integrates connections with different QoS requirements into a single stream as shown in Figure III.1. The following are typical QoS requirements of each of these connections.

Delay-sensitive connections:

Teacher's video: This requires 640 pixel × 480 pixel, 15 FPS video, which can afford moderate loss of data.

Teacher's audio: This generates 128 Kbps data and is highly sensitive to loss.

Student's or Classroom video: This connection requires 320 pixel ×240 pixel, 10 FPS video which can afford heavy loss.

Student's audio: This is again a 128 Kbps connection which is highly sensitive to loss.

Throughput-sensitive connections:

Data-transfer applications or student-controlled tools: These are typical data transfer applications which cannot afford to lose data.

Global pointer movements: When the teacher moves the pointer on the screen, the same should be visible on the other participant's screens. But this application has no stringent requirements on loss. Even if we lose some of these packets, the application QoS will not be hindered.

In such a scenario of varying QoS requirements, it is not possible to group all the applications into a single class and attempt to provide QoS guarantee. This leads to the two fundamental questions that we defined in Chapter I, namely isolation of the application and managing maintaining QoS of the application. Such knowledge of the applications is not available in the traditional communication networks. The fundamental point one should realize in designing a QoS architecture is that QoS has end user-to-end user significance, and it is quite a different view point from what is considered in designing the communication architectures. The communication architectures address the basic data transportation components between the end-systems, whereas the QoS architecture address the components needed to retain the end-user to end-user agreed upon QoS for an application. The two fundamental questions to be answered by a QoS architecture are: What is the range of applications needs it can support? and How to overcome the fundamental communication

architectural limitations to support such applications?

Supporting an application implies meeting the demands (by its nature) of the application. Let us consider a distributed collaborative application (such as IRI) and a WWW (World Wide Web) application. A collaborative application may demand for the group management of the resources across multiple connections in the application and synchronization between different connections in an application. A WWW-like application needs an architectural support for dynamically varying traffic patterns. Such demands from the applications expect certain services from the QoS architecture. For example, the collaborative applications requirements demand group resource management capabilities, and synchronization demand group allocation and group management capabilities for such applications. The requirements of WWW ask for a dynamic QoS specification and resource management. Hence, the QoS architectural solution should be scalable to different applications. Also, since the QoS is end-to-end and the traffic of these applications are carried via many networking technologies, the solution should also be flexible to accommodate multiple networks.

The QoS guarantee is a global issue and will be defeated if any one of the intermediate networking component fails to guarantee QoS. The QoS architecture used in different communication architectures are different due to their fundamental service specifications. For example, ATM is designed to be a connection-oriented protocol suite, and hence the QoS architecture developed in such an environment is also designed to accommodate individual connections, whereas IETF's IP networks do not have a concept of an end-to-end connection. As a result, the individual packets can traverse any existing route between the end-systems. When an application's communication path is traversing through such varying service-providers,

it is essential to separate the service-provider dependent solutions from the end-to-end requested solutions. In our proposal, QUANTA, we have developed one such architecture to address the QoS guarantees.

Many solutions addressed these two fundamental issues. It is known to the research community that to make use of these QoS solutions in the real-world they should satisfy the following properties — scalability of the solutions to accommodate new applications, interoperability between solutions, fairness of the solution across different service classes and among the connections in the same class, efficiency of the solution as compared to the other existing solutions, and the cost of the solution in terms of the protocol overhead. In the following discussion, we consider each of the issues, briefly outline the solutions provided to them, highlight the properties to be addressed in these issues, analyze the existing solutions from the property's point-of-view, and present how our solution addresses these issues.

2 Isolation of the application

QoS definition and provision is individual to an application. To manage and monitor its requested QoS (apart from other reasons such as connection management), an application needs to be isolated, in the network and on the host. The effort to isolate an application turns out to be a larger problem as we integrate applications with different QoS requirements, as the number of connections increase, and if there exists an inter-relation between connections (one such example is IRI).

To subdivide the isolation problem, it has become a common practice to classify the applications into different service classes and manage them as a single entity. Even after classification of the applications, it is necessary to maintain the identity of an application (with one or more connections) to monitor the QoS and

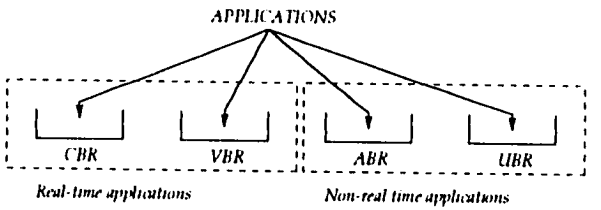
to penalize the misbehaving applications. Hence, we divide the isolation issue into two major sub-issues, namely, classification of the applications and identification of the applications.

2.1 Classification of applications

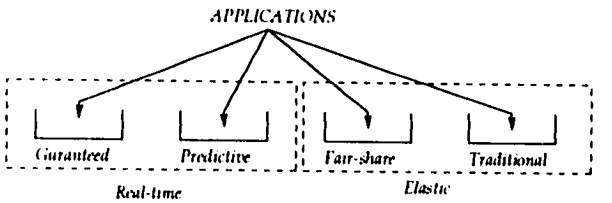
The basic idea behind classification is to group like-applications and attempt to provide their requested QoS. The fundamental question that needs to be answered by any classification scheme is: Does this classification encapsulate the present and the future applications? The same necessity can be viewed from two different angles, namely, Is this classification scalable to all the applications? and Can it completely specify an application's requirements? Some other questions are: Is it easy to find the appropriate class from the application specification? and (since the QoS need to be maintained end-to-end) Is it possible to find an equivalent class when there is a transition between the QoS architectures? We try to see the exiting solutions from the above questions point-of-view.

The two well-known classifications of applications are the ATM Forum approach and the IETF approach. Other implementations such as Tenet suite-2 [29] and QOS-A [45] fall into one of these classifications. Figure III.2 depicts the classifications provided by the three approaches we discuss in this section.

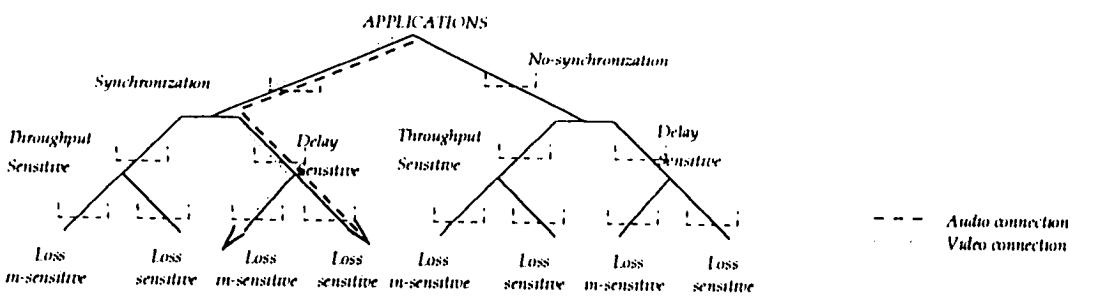
The ATM Forum proposes a flat classification (refer to Figure III.2.a) of the service classes it supports depending on the type of traffic the applications generate, timing recovery requirements in the applications, and the type of connections required (connection-oriented or connection less). The ambiguity involved in matching a service class to an application [25] shows that this classification is not complete. One such example is transferring MPEG-2 stream over ATM [26] where the appli-



(a) ATM classification of applications



(b) IETF classification of applications



(c) QUANTA's ripple-through classification of the applications

Figure III.2: ATM Forum, IETF, and QUANTA's approach to classification of applications

cation has multiple choices of AALs to use, but none can provide a solution to all the issues involved in supporting this application. We feel that much work needs to be done to make this technology scalable to different applications.

The IETF tries to provide another flat classification scheme (refer to Figure III.2.b) for the services it can support [27]. Let us consider two examples where this model fails: one is IRI and the other example is (as mentioned in [25]) where in one service class we have priorities among multiple connections. In IRI, let us say that there are two different streams for audio and video. These two fall into different classes namely guaranteed (for audio) and predictive (for video). Let us say we need to synchronize these two streams at the receiving end, which is a necessary requirement. In the current IETF model, it is not possible to specify and provide such a request. The second example is related to mixing connections with different priorities into the same class. It is again not possible to specify such requirements in the application flow [28] specification.

Other groups such as Tenet, TIP [30] proposed a QoS architecture for carrying real-time traffic. These suites do not provide solutions to classify other applications or provide a clear definition of the merge of these applications into their architecture. Because of this, we assume that they intend to use different protocol architectures to different class of applications. This method of using different protocol suites for different classes of applications aggravates the problem of maintaining a relation between related-connections, which is very common in collaborative environments.

We in this work propose a new classification mechanism called *ripple-through classification* in which we approach this problem from the application's point-of-view. The criteria used in our classification is to uniquely represent an application

(with one or more connections), whereas the other approaches only identify a connection. Any application has certain requirements of the QoS parameters, such as loss, throughput, and delay; and when we consider collaborative applications, the other requirements such as synchronization will come into picture. As shown in the Figure III.2.a or III.2.b, instead of the data related to a connection falling into a container called a service class, in our classification the data ripples through different QoS parameter containers (refer to Figure III.2.c). The main difference between our approach and the other above mentioned approaches is that we use QoS parameters to classify applications rather than service classes (where a set of QoS parameters are grouped). This approach provides high flexibility and a stream-lined approach to providing QoS guarantees as we discuss below.

We use the following terms to explain the ripple-through classification and an implementation mechanism shown in Figure III.3.

A graph is an organization of different QoS parameter links.

A QoS parameter link (link) is a container (or a scheduler) where necessary action (related to that QoS parameter) is taken on the packet entering into that container.

Head-end is the point where the data packet enters the graph.

Tail-end is the place where the packet leaves the graph.

A path is the set of links a packet takes between the head-end and the tail-end.

To demonstrate our classification we consider the two applications from IRI, the teacher's video and the teacher's audio (refer to Figure III.1). The video connection can afford losses, whereas the audio connection cannot afford losses. Also

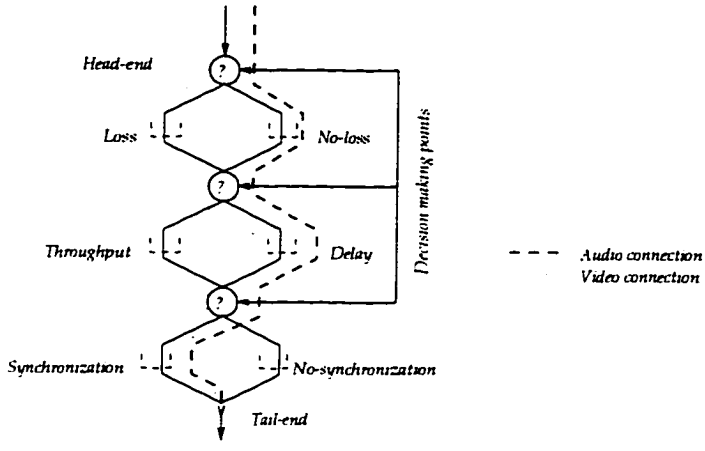


Figure III.3: Quality of Service Traversal Graph (QTG)

these two connections should be synchronized. We traced the requirements of these two connections in Figure III.2.c. If our classification is implemented as shown in the figure, these two connections take different paths in the QoS graph before reaching the tail-end. Hence, they may not be synchronized when they are transmitted. To compensate for such scenarios we should carefully consider organization of these QoS traversals through the classification graph. One such example is given in Figure III.3. Here we invert the classification graph and insert decision points after considering each connection for a specific QoS parameter. We call this inverted classification graph a QoS traversal graph (QTG).

In the loss link, data can be dropped depending on the status of the route the current packet is going to take[†]. This technique provides a clear methodology to drop packets from different connections, unlike the ambiguity in the other approaches such as the IETF approach [25]. Because of the clear definition of dropping packets

[†]Since dropping of packet takes place depending on the path the packet is going to take, the QTG needs to be implemented after the routing decision is made in IP.

and since it is done at the head-end of the QTG, the overhead of scheduling the packets which might be dropped at the tail-end will reduce. Here we make the scheduling of the data, which are well-understood in the studies of [46] and [29]. If we need synchronization between multiple connections they can be group scheduled by synchronization link of the QTG; otherwise the data can be sent directly to the network interface. For the connections that need to be synchronized, we can use some techniques like minor-major synchronization points (similar to that which are used in the OSI session protocol). To explain this with the previous example, the video connection goes through loss link of the graph and continues to the next decision point; and the audio goes through the non-loss link of the graph before reaching the decision point. Both the connections take delay-sensitive link of the graph, as they both belong to this class. Once they are scheduled, they are sent to the synchronization-link of the graph to synchronize the two streams before sent out onto the network.

2.2 Identification of the application

In the traditional networks, a connection is identified by the source destination address pair[‡]. By introducing QoS of an application, it is essential to identify the same application not only from the connection point-of-view but also from its QoS point-of-view. Hence, in the future QoS guaranteed networks an application is identified by the <Connection identifier, QoS identifier> tuple. In our discussion we only concentrate on the QoS identification. This identification helps the end-system protocols and the network in maintaining and managing the application and its QoS. The main task of a QoS architecture is to retain the QoS identity of an application

[‡]An address in IP networks is the IP address + the application port number

end-to-end. As we will see in the following discussion, this is not an trivial task when an application traverses through different networks with incompatible QoS architectures. Also, this identification needs to be flexible enough to accommodate varying application behaviors.

The ATM Forum approach to the problem of identification is using static establishment of a connection before the data transfer takes place [47] using the application requested QoS. They use the Q.2931 signaling protocol to establish a QoS guaranteed connection between the source and the destination(s). A connection is accepted depending on the traffic characteristics of the application[§], the requested values of QoS parameters[¶], the requested QoS class[‖] and the current state of the network resources. Algorithms are developed for connection admission (CAC - Connection Admission Control) and for connection monitoring (GCRA - Generic Cell Rate Algorithm). The connection is retained until it is deleted by the communicating parties. Once the connection is established, it is very involved to change the characteristics of a connection. This suggests that it is either difficult to adjust the resources to the dynamics of an application or (if we reserve more resources) we may be wasting resources. The database maintained at every switch is on a per-connection basis, and it grows as the number of connections increase. Also, the

[§]ATM Forum represents an application traffic characteristics with the help of Peak Cell Rate (PCR), Sustainable Cell Rate (SCR), Maximum Burst Size (MBS), Minimum Cell Rate (MCR), and Cell Delay Variation Tolerance (CDVT) [48].

[¶]ATM Forum defines the following QoS parameters to represent the QoS requirements of a connection: Peak-to-Peak Cell Delay Variation (CDV), Maximum Cell Transfer Delay (Max CTD), Mean Cell Transfer Delay (Mean CTD), and Cell Loss Ratio (CLR) [48].

[‖]ATM Forum has the following QoS classes: Constant Bit Rate (CBR), Real-Time Variable Bit Rate (RT-VBR), Non-Real-Time Variable Bit Rate (NRT-VBR), Unspecified Bit Rate (UBR), and Available Bit Rate (ABR) [48].

connection establishment time is more.

Other connection-oriented reservation protocols include ST-II [15] developed for Internet and Real-Time Channel Administration Protocol (RCAP) [49] developed for Tenet protocol suite. Both these protocols use the same philosophy of specifying the traffic parameters and establishing a connection with the specified QoS parameters. Since these protocols are developed for point-to-point connections, they have well understood the problem of maintaining a database for a multicast connection. Also, since Tenet protocol suite was developed for real-time applications in mind, only real-time related traffic and QoS parameters** are used in RCAP [49]. These specifications do not extend for other non-real-time applications.

IETF takes a slightly different approach to the issue of identification to provide backward compatibility with the existing versions. In IPv6 [50], new fields are provided to identify a connection (and its characteristics) with the help of a field id and priority fields. The field id in association with the source IP address uniquely identifies a connection. The difference between the ATM Forum approach and the IETF approach is that in IPv6 a cache entry is maintained for every connection, and it is deleted if the connection is idle for 6 seconds. This is to provide dynamism to the application characteristics and with the assumption that the Internet connections lifetime is very less. The priority field in the IPv6 packet is used to group connections into different service classes. Though this classification is not complete, [50] provides typical classification methodology by grouping well-known applications into different

**Tenet's traffic parameters are Minimum inter-message time (X_{min}), Minimum average inter-message time (X_{ave}), Average interval (I), and Maximum message size (S_{max}). Their QoS parameters are upper bound on end-to-end message delay (D_{max}), lower bound on probability of timely delivery (Z_{min}), upper bound on delay jitter (J_{max}), lower bound on probability of no loss due to buffer overflow (W_{min}).

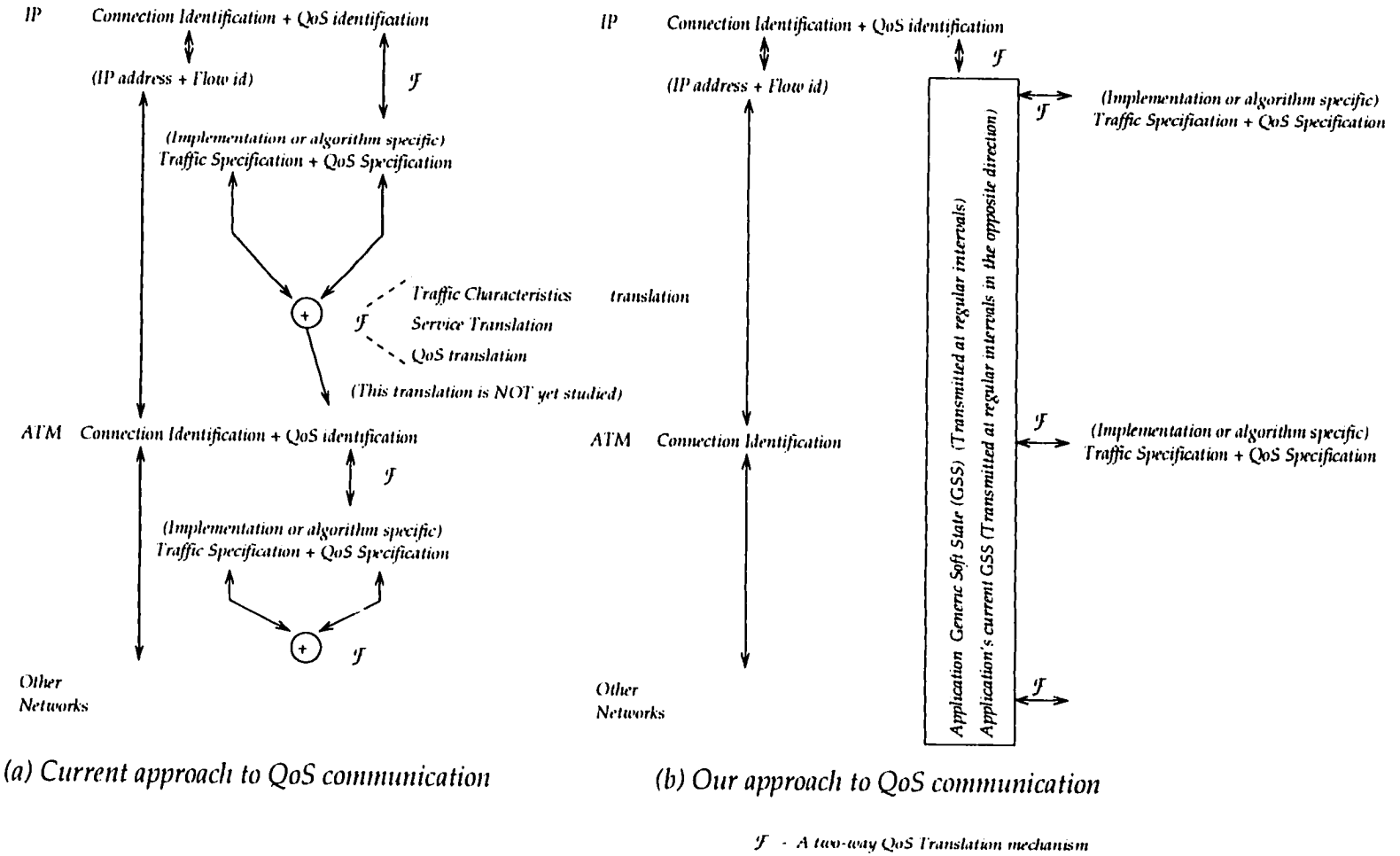


Figure III.4: QoS internetworking - other's approach and our approach

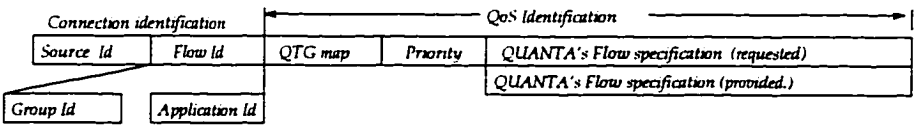
priority groups. As mentioned in the classification subsection, this method is not complete to characterize applications. Also a clear specification and communication of the QoS parameters still is not decided in this model. [50] mentions that this specification of IP version needs extension to include QoS support. It also refrains from commenting on whether the QoS reservations are done by a separate reservation protocol (such as ST-II, RSVP) or they are maintained as part of the IP protocol itself. This opens avenues to suggest ideas to manage and maintain the fields in IPv6.

In Figure III.4.a, we recapitulate the current approach taken to identify a connection and compare this approach with our proposed approach. Here we consider the teacher's video application multicasting data and assume that it is using IPv6 running on an ATM backbone network. This connection is identified by the end-system protocol-suites and the network by the connection identifier, and the QoS identifier. The connection identifier in case of an IPv6 network is its source address and the flow identifier. The QoS identification is maintained in the IP network in the form of soft state, which is the combination of the application's traffic specification and the QoS specification. Hence, in IP, the QoS identification of the video application is translated into a set of traffic and QoS parameters. These parameters are specific to the QoS reservation protocol (such as ST-II, RSVP) and are also dependent on the control parameters of the algorithms used to provide the requested QoS. When this connection uses ATM network as the back-bone transportation media, the QoS identification needs to be mapped onto the respective parameters in the ATM network as shown in Figure III.4.a. This mapping (or translation) involves determining the equivalent traffic characteristics, QoS characteristics, and the service class. This translation is performed on already translated IP soft state. These

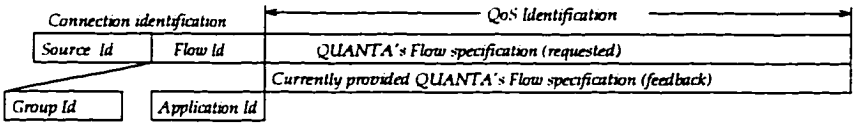
two translations need to be reversed at the destination end-system. It may not be possible to map the exact requirements of an application, as specified by the user, at different network interfaces. As we mentioned before, the ambiguity in selecting a service class to a video application in ATM itself is huge task to be handled, let alone the exact mapping of different QoS and traffic parameters from IP. It is essential to avoid this additive translation loss from the user specified QoS, as the application traverse in the network.

To alleviate the above problem of QoS translation loss, we propose separating the QoS identification from QoS managing and QoS monitoring tasks as shown in Figure III.4.b. In our approach we send the originally specified QoS (Generic Soft State [GSS]) identification end-user to end-user and let the end-system protocol suite and the networking paradigms extract their protocol specific QoS and traffic parameters. The advantages of this architecture are the end-to-end retention of the user specified QoS identification and reduction in the complexity of QoS translation from architecture to architecture. The elegance of this approach is that one can still maintain the capabilities of the QoS provision algorithms and the protocols in their domain. The GSS serves two purposes, one to specify and dynamically change the application characteristics, and the other is as a probe to monitor the current status of the network and the application behavior. In the later use, the generic soft state is sent in the opposite direction to the data transfer path of the application. This packet addressed as current generic soft state (CGSS) can help as a feedback packet to control the resources in the forward direction to maintain QoS.

Figure III.5 represents the preliminary format of a GSS packet. This packet has two parts, one to represent the connection identification and the other to represent the QoS identification. The connection identification, which uniquely identifies



(a) Generic Soft State packet format



(b) Current Generic Soft State packet format

Figure III.5: Generic Soft State and Current Generic Soft State packet formats

a connection in an application, is divided into source identifier and the flow identifier. The source identifier is the typical end-system address, such as the IP address, where as the flow identifier is the application identifier on the given end-system. The flow identifier can represent a connection or a group of connections. This notion of group can be extended to maintain and manage a group of connections, such as the connection opened by IRI. Group identifier also enables to allocate resources on group basis and reduces the data-base overhead at the routers. The QoS identification of the GSS packet represents the QoS and traffic requirements of the connection. This contains the QoS Traversal Graph (QTG) map for this connection, priority of the connection amongst like-connections, and the flow specification (which we will elaborate in the forthcoming discussion).

3 Knowledge of the application

The next important issue in a QoS architectures is to provide necessary components to manage and maintain the knowledge of the applications on the end-systems and

in the network. The components required for the management of the application knowledge are: a QoS and a traffic specification mechanism, a QoS translation mechanism between different communicating elements in the application's communication path, a resource reservation protocol to propagate the QoS and traffic requirements of the application, and a dynamic negotiation mechanism to change the current resource allocation to an application depending on the current status of the elements in the communication path. The maintenance of the application knowledge involves the following components: an admission control component to limit the applications from sharing the resources, a mechanism to guarantee the QoS requirements of the admitted connections, and a mechanism to monitor the behavior of the applications and the status of different resources. The application knowledge management tasks contribute to the architectural components, whereas the application knowledge maintaining tasks contribute to the algorithms to provide QoS guarantees. Since we are interested in providing an architectural solution, we concentrate on the management components. The solutions provided to the management components need to meet the application service requirements.

In Figure III.6 we identify the relation between different QoS management components. The QoS translator as shown in Figure III.6 lies between the service user and the service provider, translates the user-related QoS specification, traffic specification, and service specification into their corresponding language understandable by the provider. In this work we refer to the QoS specification, the traffic specification, and the service specification as QoS specification. On the receiving side of the application, the tasks of the user and the provider reverse, and hence the QoS translation is also reversed leading to inverse translation between the service user and the service provider. Since the current QoS solutions are not inter-operating,

the QoS communication and the QoS negotiation is between the like-protocol-suite such as between two TCP/IP protocol-suites or between two ATM switches. In the following subsections, we consider each of these components relevant to the QoS architecture and discuss them in detail.

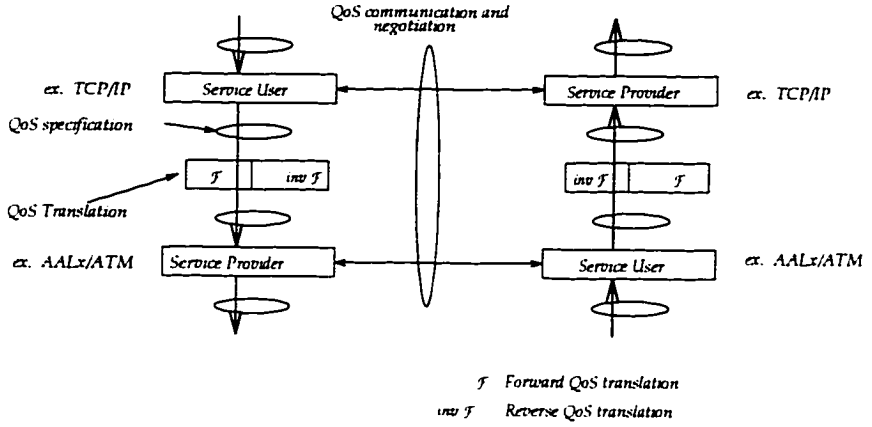


Figure III.6: Relation between QoS Specification, Translation, Communication and Negotiation

3.1 QoS specification and QoS translation

The following are the typical properties expected of a QoS specification and a QoS translation mechanism:

The specification and the translation between the user and the provider should be complete. $QoS\ Specification_{user} \rightleftharpoons^{F}_{inv\ F} QoS\ Specification_{provider}$ mapping should be one-to-one and onto. This requirement also implies that the specification needs to encompass all the current and future applications and all types of QoS specifications.

Also, the QoS specification should be service user friendly and service provider friendly, which means that the specification should be simple (i.e. easy to specify and represent) and the translation between the user and the provider should be less complex.

It is difficult to achieve the above two properties for the following reasons.

First, different organizations are involved in the QoS specification for their native network protocol-suites. These specifications differ from each other fundamentally because of the design aspects considered while developing the standards for these protocol-suites as discussed before.

Second, the most important task of inverse translation becomes ambiguous for the following reasons. Let us consider the example of teacher's video from IRI. Let us assume that the bandwidth available for this application over the network has reduced. The ambiguity comes in deciding whether this current status needs to be translated into reducing the throughput or increasing the loss. And, how does this situation translates at the application-level? Does this situation reduce the FPS, or decrease the quantization?

The first reason translates into QoS internetworking between different approaches. In QUANTA we solve this problem with the help of end-to-end GSS packets and a translation mechanism between GSS and the native protocol-suite QoS mechanism as shown in Figure III.4. The second reason implies that in addition to the triplet of <Traffic Specification, QoS Specification, Service Class>, we need to add QoS translation rules (QoS Rules) component. This has to be part of the GSS/CGSS packet as one of the QoS identification components. Figure III.7 compares the Tenet, the ATM Forum, the IRTF (Internet Research Task Force),

GENERIC QoS Specification < Traffic Specification, QoS Specification, Service Class >

TENET's QoS Specification < Limited by one class, Limited to one class, RT-class >

ATM's QoS Specification < Limited by 5 classes, Is not generic enough, (CBR, VBR, ABR, UBR, nrt-VBR)>

IRTF's QoS Specification < Limited by 4 classes, Have many limitations, (Guranteed, Predicted, Fair-Share, Best-effort) >

QUANTA's QoS Specification < Unlimited, Range-of-QoS parameters, not dependent on classes, QoS Rules >

Figure III.7: Comparison/salient features of different QoS flow specification mechanisms

and the QUANTA's approach to flow specification mechanisms. In the following text, we compare our approach with that of the IRTF's flow specification in RFC 1363 [51].

RFC 1363 is referred to by many Internet QoS-related documents, such as [52]. This RFC presents a flow specification packet as shown in Figure III.8.a and comments on why this set of fields are thought to be both necessary and sufficient by the IRTF. In this packet the traffic flow is characterized by token bucket rate, token bucket size, and maximum transmission rate. Delay is characterized by minimum delay noticed and maximum delay variation; loss is represented as loss sensitivity, burst loss sensitivity and loss interval; and finally quality of guarantee field is used to indicate the type of service guarantees that an application desires. The last field of guarantee specifies different service classes to classify applications. The three limitations mentioned in this RFC are:

> The loss model is imperfect. It is difficult and a crude way to classify an application's loss sensitivity by loss and burst loss parameters.

The minimum delay sensitivity field limits a flow to stating that there is one point on a performance sensitivity curve below which the flow is no longer interested in improved performance.

The service models are clearly not a complete set.

Version	*Maximum Transmission Unit*
Token Bucket Rate	*Token Bucket Size*
Maximum Transmission Rate	*Minimum Delay Noticed*
Maximum Delay Variation	*Loss Sensitivity*
Burst Loss Sensitivity	*Loss Interval*
Quality of Guarantee	

Version	*User-Submitted Packet Size*
R Token Bucket Rate	*R-Token Bucket Size*
R-Max. Transmission Rate	*R-Minimum Delay Noticed*
R-Maximum Delay Variation	*R-Cell Loss Ratio*
R-Loss Interval	*R-Affordable Cost*

Figure III.8: Comparison of RFC 1363 and QUANTA flow specification packets

As shown in Figure III.8.b, QUANTA flow specification packet (which is part of the GSS/CGSS packet as shown in Figure III.5) utilizes some of the arguments made by RFC1363 but with the following fundamental differences:

RFC 1363 uses hard guarantee to represent the QoS parameters and uses the Quality of guarantee to specify the requirement of soft guarantees. Since the service models are not clearly defined by IETF (as discussed in the classification section), this model will not complete the flow specification. Hence, we use range of QoS parameters instead of single values as shown in the figure.

Instead of Maximum Transmission Unit, as specified by the RFC 1363 flow specification, we use user-submitted block size as a single unit of data. The user submitted block size is the data unit submitted by the application to its immediate service provider (for example, an FTP application sending 8 Kbyte data units to the immediate protocol - TCP). This block size has more

significance in the sense of controlling different QoS control parameters as discussed in [53].

We represent loss with the help of loss ratio and loss interval instead of the three parameters used by RFC 1363. This is because these values can be controlled with the help of GSS/CGSS packet transmission.

We do not use the Quality of Service guarantee field, as the classification is provided by the QTG field in the GSS/CGSS packet.

As we mentioned before, we provide a new QoS parameter to the list, the affordable cost by the user. This could be translated into different parameters; one such cost translation is into resource cost.

We also introduce a new QoS component called the QoS Rules component, as shown in Figure III.7. We use the following mechanism to represent the rules of the application into QoS Specification packet. The user along with the requirements of the primary QoS components, Throughput, Delay, Jitter, Loss, Synchronization and Cost, provides set of rules defining the importance of different QoS components. A QoS Rule Translator is used to translate these requirements into weights corresponding to each of these primary QoS components as shown in QUANTA's packet format in Figure III.8.b. These values can be forward and inverse translated at the QoS translator between different protocol-suites.

3.2 QoS communication

QoS communication protocol (also referred to as Resource Reservation mechanisms) depend on the application communication architectures, such as 1:1, or 1:M, or M:N

communication between the source and the destination.

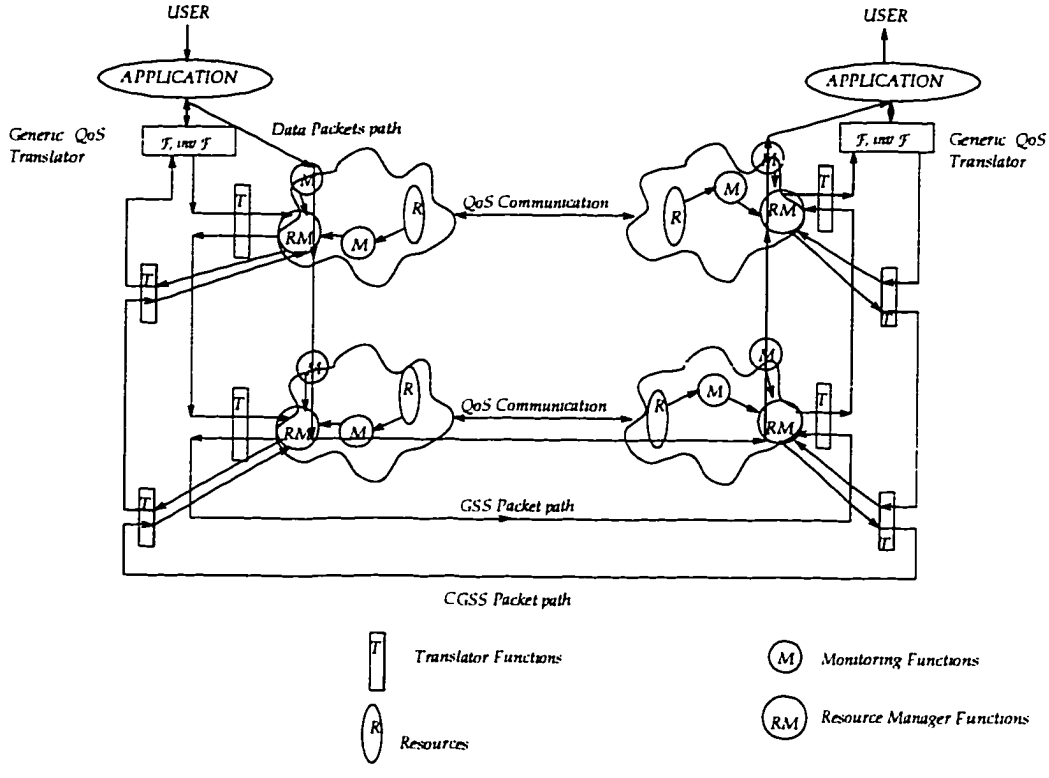


Figure III.9: Our QoS architecture

There are many QoS communication protocols proposed for 1:1 communication, such as Tenet's RCAP [31], ATM's Q.2931 [32]. The main disadvantage of such protocols is that they are not scalable to the other two-communication architectures. The scope of some of these protocols are limited by the architectural support from their data communication counterparts (for example, Q.2931 over ATM). ST-II, which was developed to manage the resources [15] in the early ages of Internet (before consolidating the idea of multicasting), makes a point-to-point source invoked reservation between the 1:M communicating parties. The disadvantages of ST-II are that it assumes that the source has complete knowledge of its M destinations

(which translates into interrupting the source when ever a new connection is added or deleted), and the reservation available to all the destinations is the minimum of all the supported connections. The above two problems are rectified in RSVP by invoking the reservations from the receiver side and by maintaining a soft state of a connection, and by using filters to scale the source emitted stream of data depending on the receiver capabilities. Because of such scalable concepts introduced in RSVP, it can support M:N communication architectures.

In QUANTA, we use the QoS communication support provided by the underlying communication architecture to make the reservations, as shown in Figure III.9. But the difference is that the GSS (and CGSS) packets are sent in-band to the data path. Whenever there is an architectural support for in-band resource communication, such as the RM cells in ATM networks, we propose to duplicate these packets into such data structures. When there is no in-band architectural support as in IP, we propose to communicate with the resource manager (RM). In networks where there is no support for resource reservation, we tunnel-through the GSS (or the CGSS) packets as they are in-band packets. To reduce the overhead per connection due to GSS packets and to allocate resources on group basis, we can extend the GSS concept to a group of connections (rather than for individual connections), as shown in Figure III.9. This opens up the area of group management and maintenance of connections.

3.3 QoS negotiation

QoS negotiation is the deliberation between the application and the resource managers and between the resource managers in allocating resources to the application. A QoS negotiation component is an important task in deciding the support for

different applications.

Different architectures considered in QoS negotiation are uni-directional, bi-directional or triangular architectures depending on the application requirements.

A uni-directional architecture include applications which mostly use broadcast and multicasting. In such architectures there need not be any comprehensive QoS agreement between the source and the destinations. These applications can also be called take it or leave it type of applications, because there is no commitment between the applications. Until now, to the knowledge of the author, only RSVP is capable of supporting such architectures.

A bi-directional architecture includes an agreement between either peer-to-peer or between service user-to-service provider (refer to Figure III.6 for the definitions of these terms). In such applications, when peer-to-peer agreement is supported the service provider is not allowed to change the requested QoS. Vice versa is also true for service user-to-service provider. Such applications include the multicasting or broadcasting with resource reservations only with the network, where both the source and the destination come into some QoS agreement with the resource managers on the communication path.

In a triangular architecture, both the peers and the service provider come to a QoS agreement. There are many QoS communication protocols in literature which support such architectures. For example, source negotiation initiated ST-II and receiver negotiation initiated RSVP are well known in the Internet community ([52]). ATM's Q.2931 is another example for the triangular negotiation architecture. Since in QUANTA we use the underlying QoS communication architecture, the proposed solution is independent of the ne-

gotiation mechanism, as long as the GSS and CGSS packets can be generated from both the peers and the service provider (Resource Manager).

The negotiation mechanisms between the peers and the service provider include: one-time negotiation, complete renegotiation, best-effort negotiation, and semi-automatizated negotiation.

The one-time negotiation mechanism is used by ATM Forum (for CBR, nrt-VBR, and VBR type of traffic), which expects the application to characterize its traffic pattern and the QoS requirements before the connection is established and confirm to it until the the application is closed. When the application needs to change its QoS requirements, it needs to re-establish a new connection with the network.

A complete renegotiation involves interrupting the end-to-end components whenever a change of state occurred in the network. This calls for more elaborate procedures for negotiation. No existing solutions provide complete renegotiation mechanisms.

The best-effort service provided by both the IETF and the ATM Forum do not involve the application in the case of highly deviated QoS parameters. The ATM Forum approach, though it is in the developmental stage, tries to guarantee only loss for the best-effort class (ABR class) application [54], which may not a primary criteria for an application using ABR class.

In the semi-automatization mechanism used for QUANTA, we use a combination of the range of QoS parameters, weights (to identify the importance of different QoS parameters), and feedback from any of the resource managers to degrade a stream. We try to adjust the QoS values inside the network

within the specified range of QoS parameters, when it is impossible to provide such guarantees a CGSS is sent to the user application for QoS modifications. Since, dynamic nature of the applications are supported by QUANTA, an application can react to the current state of the communication components and send a new GSS packet requesting for the change of QoS.

As observed in the negotiation mechanisms discussion, feedback is an important issue to change the current state of the application requests to accommodate the congestion on the network. Current feedback mechanisms are loosely controlled in the sense that the reaction time is in the order of round-trip-time between the communicating nodes. For example, ATM Forum uses RM (Resource Management) cells end-to-end to control the data rate generated by the best-effort applications. This could lead to complicated algorithms for the dynamic negotiation and can create oscillations in the network traffic if the algorithms are not properly devised. To alleviate this problem we let the intermediate resource managers exchange CGSS packets among themselves to react quickly to the changing state of the network.

4 Summary

In this chapter we identified the issues involved in a QoS architecture, discussed the existing solutions to these issues, and provided our approach and solutions to these issues.

These issues include isolation of the applications and managing the knowledge of the applications. The issue of isolating an application is sub-divided into classification of the applications and identification of the applications. Addressing these issues we propose a ripple-through classification mechanism and a Generic Soft

State (GSS) identification mechanism. To manage the knowledge of the applications we propose different QoS components, such as a GSS negotiation mechanism, GSS communication mechanism and GSS monitoring mechanism (such as GSS Relays and GSS Agents). In the next chapter, we present the design methodology for the proposed architecture.

CHAPTER IV

DESIGN AND EVALUATION METHODOLOGY

Details: We think in generalities: we live in details.

— Alfred North Whitehead

In the previous chapter, we discussed the fundamental QoS architectural components needed and our proposed solutions to the issues raised. These QoS components cover the entire path from end-user to end-user. We group these QoS components into implementation components: the user-level implementation components and the protocol-level implementation components. Figure IV.1 describes the relation between different QoS components and the implementation components. In the following sections, we will consider each of these components, identify their subcomponents, describe the tasks of each of the subcomponents, and provide design details.

The application-level components, which are presented in section 1, include a TLI-like QoS interface to the application (which apart from managing the connections, maintain, manage and renegotiate QoS of different connections in an applica-

tion) and a resource reservation daemon to manage the resources of the end-system. In section 2 we present the protocol-level components, which include a Generic Soft State (GSS) component, a resource management component, and a QoS provision component. Portion of QUANTA is implemented under Solaris, that is, both host and remote workstations running a distributed application must use Solaris; the methodology for evaluating the effectiveness and cost of running applications under QUANTA is given in section 3. The design is summarized in section 4.

1 User-level implementation components

The user-level components include a User-level QoS Library to provide a QoS interface to the end-user applications and a Resource Management Daemon which is an interface between the host and the network resources. It keeps track of the current status of the end-systems' resources and reserve and allocate these resources to different applications.

The functionality of these components is grouped into subcomponents to reflect $\{X\}$-dependent and $\{X\}$-independent modules. The 'X' could be an application, a communication protocol, the host-operating system or the underlying network. For example, in Figure IV.1 the QUANTA user library is independent of the application or the underlying communication protocol. Similarly, the core stub in the daemon is independent of the host operating system, the user library, or the underlying communication protocol. The access between these subcomponents is through the message specifications in the header files mentioned at the respective interfaces in Figure IV.1.

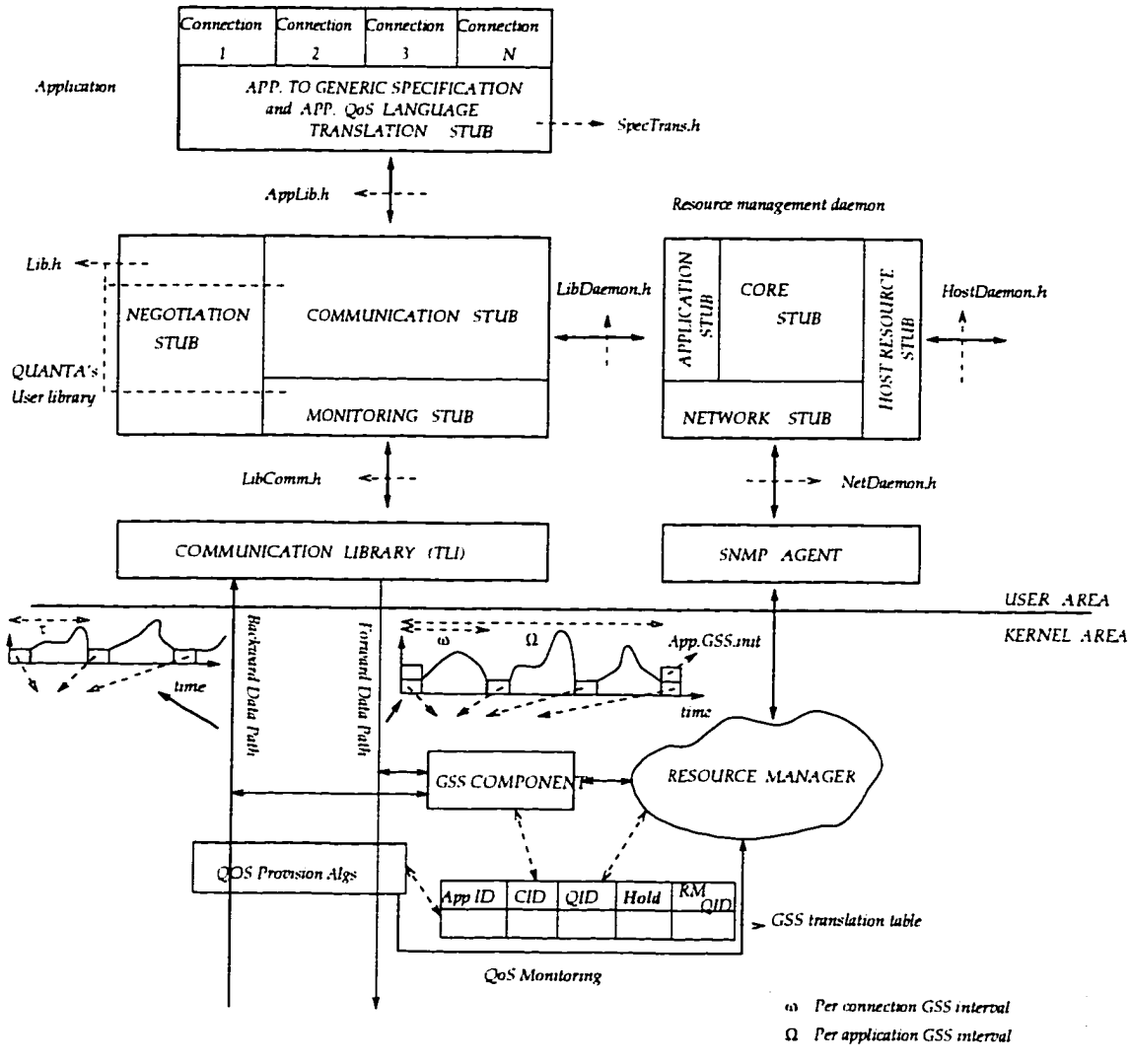


Figure IV.1: Implementation components in QUANTA

1.1 QoS user interface

The QoS interface provides a TLI-like communication interface with QoS negotiation capabilities to an application. The major tasks of this interface are: application dependent tasks, application independent and protocol independent tasks, and protocol dependent tasks.

The application-dependent tasks are:

QoS Specification: Using this interface an application can specify QoS parameters which are relevant to its behavior. For example, a video connection can specify the number of frames per second, the quantization, etc.; an audio connection can specify the precision, the number of channels, etc. We also specify the relationship between connections within an application.

QoS Translation: The application-dependent QoS specification is translated into a generic QoS format [55] using this translator. It also provides an inverse translation from the generic to the application QoS.

QoS Language Translation: We provide an interface to the application to specify the importance of different QoS parameters and translate them into QoS weights.

QoS renegotiation primitives: Some primitives are provided to renegotiate the initially allocated QoS to an application and to a connection.

The QUANTA's user library supports the application independent and protocol independent tasks. They are:

QoS Reservation: An application can request to reserve its expected generic

QoS with its local resource reservation daemon. It sends a GenericQoS.Req message and expects a response in a GenericQoS.Res message.

QoS Maintenance of an application and a connection: Once the reservation is made, the interface needs to maintain the reservation during modification of the QoS requests or while adding a new connection to an already existing set of connections.

QoS Communication: The locally agreed upon QoS needs to be communicated end-user to end-user. We use GSS.Init and GSS.Mod to communicate the application and its connection's QoS requirements, when they are initialized and modified respectively.

QoS Monitoring: The receiving data is passed through a monitoring stub to verify that the connections are within the requested QoS.

QoS Semi-automatized renegotiation: With the help of GSS.Reg, we inform the end users of the current state of the network and the end-system components to allow for dynamically adjusting a connection's behavior. The application is involved in this dynamic negotiation only when the QoS reaches outside the range of the negotiated request.

QoS Relinquish: When a connection or an application is closed, the resources occupied by them are relinquished. Local resources are relinquished with the help of the daemon and the network resources are relinquished by timing out the allocation for a connection. In response to a relinquish, a GenericQoS.Rel message is sent to the daemon and the daemon sends back a GenericQoS.Rel.Res message to acknowledge the relinquish message.

The protocol dependent tasks, which are part of the communication library as shown in Figure IV.1, include:

Registration of the connection: In this phase we register a connection with the requested protocol-suite, bind the local end-point and the destination end-point to an address, and set appropriate protocol-dependent control parameters such as sender and receiver window sizes depending on the connection's QoS request. For the relation between the QoS and the TCP, UDP or AAL5 protocol control parameters refer to [53].

Connection establishment, data transfer, and disconnection: Here we establish protocol dependent connections between the end-points, transfer data between (among) them and disconnect (orderly release, abort or close) them.

The core of the QoS management of an application lies in the QUANTA's user library. We divide the tasks in this library into six different design phases. They are the registration phase, the passive connection phase, the active connection phase, the data transfer phase, the dynamic renegotiation phase, and the GSS management phase. The last three phases occur during the data transfer phase of a connection. For the sake of easy specification and tractability of the interface code, we divide the data transfer phase into the three mentioned phases. Also note that there is no specific disconnect phase mentioned in the above phase specification. A connection may be closed for many reasons from any of these phases and therefore this task is part of all the above specified phases. Whenever a connection is closed, its resources are relinquished. We outline these phases below: for extensive details please refer to [56].

In the **registration phase** an application and a connection registers itself with its local end system. Here, we verify the requested QoS with the local re-

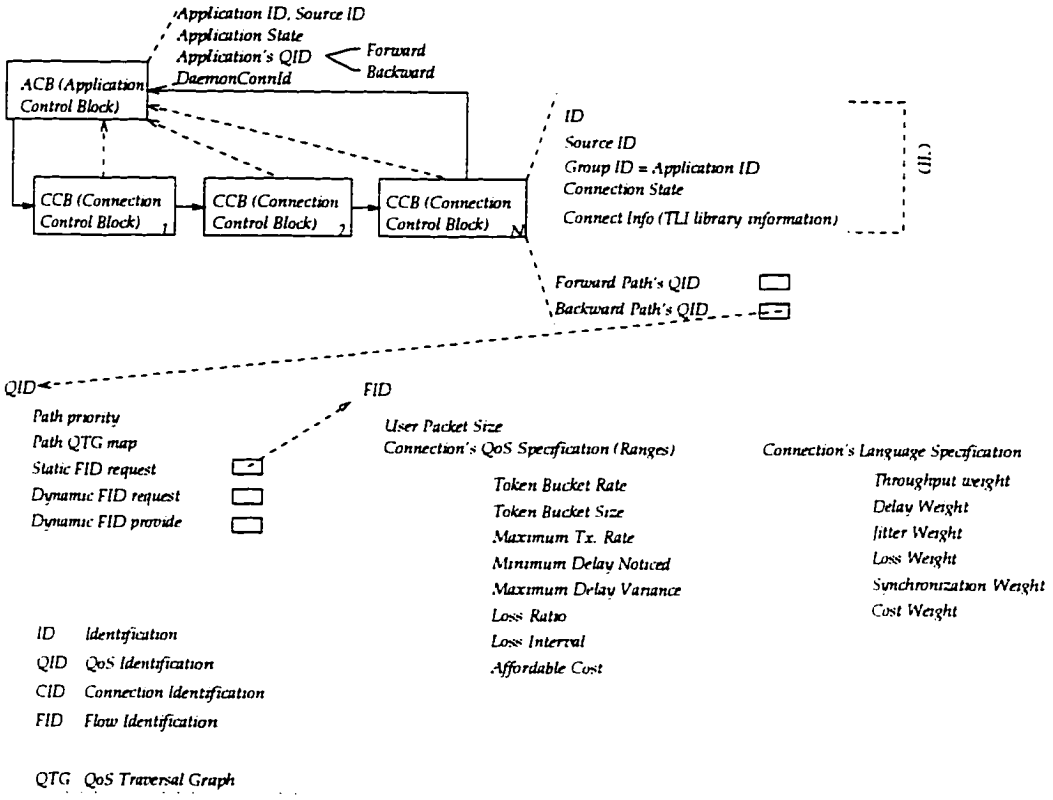


Figure IV.2: Application and connection control block structures for the QoS user interface

source daemon. We also bind the connections with their local addresses. When an application is registered, as shown in the Figure IV.2, for every application an Application Control Block (ACB) is maintained at the interface. Each ACB contains the application's connection identifier, which is <Application ID, Source ID>*, the application state and an applications QID which determines the QoS requirements of the application. This QID is specified for the outgoing connections (Forward), and for the incoming connections (Backward) into the application. The application can reserve resources for its connections a priori or update the resources as and when a new connection is added in the application. The daemon connID field contains the application-daemon connection related information to contact the local resource management daemon. Similar to ACB, each connection is allocated a Connection Control Block (CCB) when it is registered. When a connection is accepted its CCB is linked with the ACB as shown in the Figure IV.2 and if the application did not reserve resources the QID in ACB is updated. Again, every connection is represented by a <CID, QID> tuple. A CID contains <ID, Source ID, Group ID> to uniquely identify the connection and its parent application (Group ID = Application ID), connection state, and connection information which contains the protocol-related connection information. A QID contains the priority of the connection among the other connections in the application, the path QTG and the set of flow identifiers (FIDs). A static FID request is used to store the initial QoS specification of the connection, a dynamic FID request is used to maintain the current QoS request due to the dynamics of the network, and a dynamic FID provider is used to maintain

*The application ID is a unique ID (on a given host) provided by the resource daemon at the time of registration. The source ID is the local host address (could be an IP address or an E.164 address). Hence, the tuple <Application ID, Source ID> uniquely identifies an application in the network.

the current QoS provided to the connection. An FID is a combination of user's packet size, connection's QoS specification, and connection's language specification. For more details on these parameters, please refer to [55]. An application or a connection has to perform registration before invoking any other connection-oriented tasks.

In the **passive connect phase** an application waits for an incoming connection request. When a connect request is received, it verifies if the requested QID can be allocated; if yes, the connection is registered, the local resource status is updated and then a connection accept is sent; or else, the interface responds with the best it can provide and keeps the required resources on hold. These resources on hold are timed out if they are not allocated after certain time. In the **active connection phase**, a connection establishes a physical connection between the end-systems and then sends a GSS.Init (refer to Figure IV.3). The GSS.Init packet request resources for the forward or the backward connection depending on the direction of the data flow. The response, CGSS.Init packet, is analyzed against the request, and if the response is agreeable, the connection is registered or else it is disconnected. A summary of the direction of different GSS messages is shown in Figure IV.1.

During the **data transfer phase**, both on the transmitting side and on the receiving side, data is monitored to check if it is within the negotiated limits. If a connection exceeds the limits, it is penalized immediately by dropping the packets. A **dynamic renegotiation phase**[†] is invoked from a transmitter of data, which

[†]The dynamic renegotiation has two different connotations, one when a connection itself wants to change the QID and the other when a connection needs to react to dynamically to the state of the network. We provide solutions to both of them. In the dynamic renegotiation phase we provide a solution to the first concern and in the GSS management phase we address the second concern.

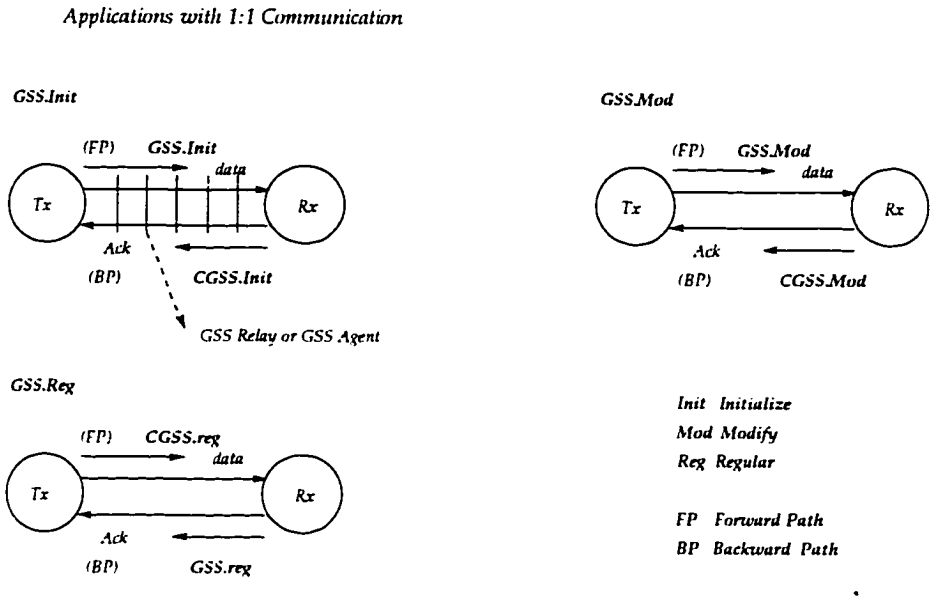


Figure IV.3: GSS/CGSS message types and their origination in 1:1 communication

submits a modification of the earlier requested QID. This new QID is verified on the local system and then is sent in the forward data path to the receiver in the GSS.Mod packet as shown in Figure IV.3. The response to the GSS.Mod, the CGSS.Mod, is analyzed at the transmitter which registers the new QID of the connection if the response is acceptable. In the **GSS management phase**, a receiver receives a GSS.Init and a GSS.Mod packet in the incoming data connection and the receiver generates CGSS.Init and CGSS.Mod in response to them. Also, during this phase a receiver sends GSS.Reg packets at regular intervals (ω, as shown in Figure IV.1) summarizing its status on the backward connection (each connection also carries the QID summary of the application at regular interval of Ω, as shown in Figure IV.1); the transmitter sends a response to the GSS.Reg packet in a CGSS.Reg packet. A detailed handling of the GSS.Reg and CGSS.Reg for a point-to-point and point-to-multipoint connections is discussed in the protocol components section.

We test the correctness of this QoS Interface with the standard protocol verification methodology. We will have this interface between an upper tester and a lower tester. An upper tester generates an end-system connection-oriented primitives which will pass through the interface to the lower tester. The lower tester emulates the responses of the other end-system connection. The test cases are chosen such that the interface goes through all the internal states. We note the responses of the interfaces and manually verify them if they are correct.

1.2 Resource Management Daemon

The resource management daemon keeps track of the current status of the resources on the host machine and the resources allocated to the application. The daemon has four different threads (or stubs) as shown in Figure IV.4, which can be divided into generic QoS-dependent stubs, an operating system dependent stub, and the network management-dependent stub. The application stub and the core stub fall into generic QoS-dependent stubs, because they view QoS in terms of generic QoS parameters. The resource stub is dependent on the Operating System of the host machine[‡]. The network stub collects the statistics (or Management Information Blocks [MIBs]) from network management elements such as an SNMP agent and provide this information to the core stub. We present the tasks of these four stubs below.

Application Stub: An application negotiates with this part of the daemon to check if the requested QoS is supported on the host machine. This is one user-

[‡]Since we do not provide a solution to reserve process-dependent resources on the host machine, this stub can only supply the current status of the host resources. However, if a reservation mechanism such as [57] is available, then this stub can be instantiated to reserve resources to an application.

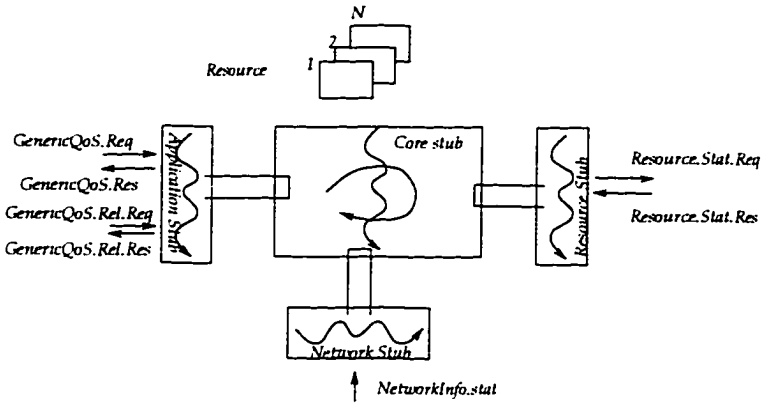


Figure IV.4: Tasks of daemon component

level thread or a process which can take the GenericQoS.Req and responds to it with GenericQoS.Res. The scheduler in the stub will take the request message and place it in a queue which is served by the "Core Stub" which has global knowledge of the resources (as shown in Figure IV.4). The response to the request is sent back to the scheduler, which relays it to the application. When an application relinquishes certain resources, it sends a GenericQoS.Rel.Req to the stub. This message is passed to the "Core Stub" and a response to the relinquish is sent in the GenericQoS.Rel.Res message.

Resource Stub: This stub talks with the system resources to obtain the current state of the machine and passes this information to the "Core Stub." This stub keeps track of the level of utilization of the designated resources and sends the information to the "Core Stub" to update the records of the corresponding resource. The messages sent to the "Core Stub" are in the <resource, occupancy> format. It invokes a query status request to a particular resource

at regular designated intervals[§].

Feedback Stub: This stub talks with the QoS manager in the kernel to get the current state of the network and the protocol resources to provide the applications with the most recent information. It submits the information of the network resources at regular intervals. It can also emulate the utilization of different kernel resources such as buffers. This gives the QoS resource manager the flexibility to constrain the availability of the resources at its discretion.

Core Stub: This stub maintains the current status of the resources. A scheduler is used to poll the three queues from the above three stubs (as shown in the Figure IV.4). Depending on the queue from which a message is extracted, we choose the message format. Its responsibilities include:

Maintaining the resource status. The resource status is translated into generic QoS parameters and is maintained in a table. The table contains an entry in <Resource name, details, limit, occupancy, QoS Equivalent> format for each of the kernel resources.

Sending a poll message to the resource and the feedback stubs to update the current state of the resources.

Translating the resource requests into supported generic QoS parameter values.

Responding to QoS requests from the application stub.

[§]The resource stub is Operating System dependent, we provide a resource stub written for Solaris 2.4. Currently, this stub communicates with the rpc.stat daemon, which keep the status of the kernel resources.

2 Protocol-level components

The protocol-level components include: a GSS component which is categorized as (depending on the nature of processing the component does on the GSS packet) the transmitter, the GSS relay, the GSS agent and the GSS receiver; a QoS Provision Algorithms Component which will provide the requested QoS to a connection looking at the GSS packet and registering that information in the local data structures: and a Resource Manager Component which has the current status of the protocol resources and assists the GSS Component in modifying an existing or a new connection's QoS.

2.1 GSS component

The GSS component extends from the end-user to the end-user. In Section 1.1, we defined different GSS message types and their use during the communication. Here, we identify the tasks of GSS messages and the involvement of different implementation subcomponents on these messages.

A GSS message is used to accomplish the following tasks:

QoS specification: It specifies the generic end-to-end QID of an application or a connection within the application. This GSS message retains its basic data structure as it traverses through networks with their native QoS architectures.

QoS translation: When the GSS packet travels through different QoS architectures, the Generic QID used by the GSS packet needs to be translated into the QoS architecture's native format; the same needs to be done from the QoS architecture QID format to the Generic QID. As shown in Figure IV.1, a GSS component talks to the resource manager or to the resource daemon (in the case of the application-level component) to inform the connection of the

current status of the resources.

QoS communication: A QoS communication methodology is required to carry the QID end-to-end. In 1:1 applications, as shown in Figure IV.3, the flow of GSS messages is fairly simple. Whereas in 1:M applications, the QoS architecture needs to treat different end-systems and different connection subtrees separately. We will discuss these two types of communication in detail below.

QoS negotiation and renegotiation: This task is performed when an application end-user changes its QID characteristics dynamically and when the resource state of the network changes. The major property of a QoS architecture is to react quickly to such changes. Hence, the transmitter needs to be informed about this change as soon as possible.

The last two tasks imply that a GSS message should also be generated from the GSS components in the network. If every element in the QoS architecture can generate a GSS message the overhead of the protocol increases. Hence, apart from the transmitter and the receiver, we designate some components in the communication data path as GSS agents, as shown in Figure IV.5, who can act as a sink or a source of GSS packets. In contrast to a GSS agent, a GSS relay can only relay a GSS message. So, a transmitter and a receiver can specify, negotiate, and renegotiate QID; a GSS agent can translate, communicate, negotiate, and renegotiate QID; and a GSS relay can translate and communicate QID.

The GSS component interacts with the other two protocol-level components (the resource manager and the QoS provision components) through a GSS translation table (see Figure IV.1) and interface messages. At the arrival of a GSS.Init message, the GSS element checks the current state of the resources in the table

and modifies the GSS.Init message to the best value provided by the local resource manager. These resources are kept on hold for certain duration before they time out if there is no CGSS.Init, and they are allocated with the reception of CGSS.Init. The same procedure is used with the reception of GSS.Mod message. For 1:M applications the GSS agent will duplicate the request and send it in all the outgoing connections for that application. Processing of GSS.Reg is more involved, because it is receiver invoked. In the following paragraphs, we explain the processing of a typical GSS.Reg message at a GSS component.

Basic GSS algorithm

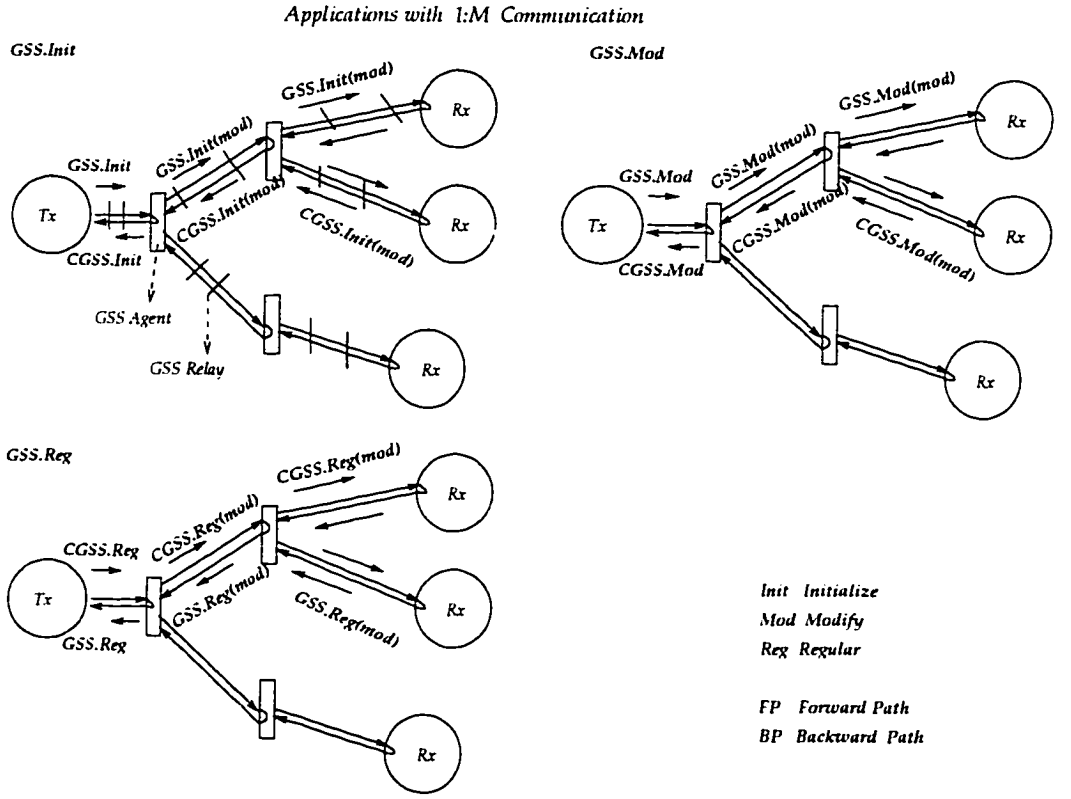


Figure IV.5: GSS/CGSS propagation in 1:M communication

```
RECEIVED GSS.Dynamic.Request (1:1 communication)
        GSS.Dynamic.Request = SEND_GSS_REQUEST (GSS.Dynamic.Request)
        if( Position == GSS_AGENT OR GSS_RELAY)
           Forward GSS
        else if ( Position == TRANSMITTER )
           RESPOND_GSS (GSS.Dynamic.Request)
```

In 1:1 applications (as shown above), when a GSS.Reg message is received at a GSS component, the current state of the resources that can be allocated are obtained from the algorithm SEND_GSS_REQUEST. If the GSS component is a GSS agent or a GSS relay, the message is forwarded to the next GSS component, or else if the GSS component is the transmitter it responds back to the GSS.Reg with a CGSS.Reg in the forward data path.

```
RECEIVED GSS.Dynamic.Request (1:M communication)
        if( Position == GSS_AGENT )
           RESPOND_GSS (Local.Dynamic.Provide)
           Wait for X-units of time to accumulate the other requests from the subtree
           GSS.Dynamic.Request = Summary of all GSS.Dynamic.Requests
        GSS.Dynamic.Request = SEND_GSS_REQUEST (GSS.Dynamic.Request)
        if( Position == TRANSMITTER)
           RESPOND_GSS (GSS.Dynamic.Request)
        else if( Position == GSS_AGENT OR GSS_RELAY)
           Forward GSS.Dynamic.Request
```

In 1:M applications (as shown above), when a GSS.Reg message arrives at a

GSS component and if the component is a GSS agent, a response is sent back with the QID agreed in the previous GSS cycle. Also at the GSS agent, the GSS.Reg messages are accumulated until all the requests are received from the connection subtree or wait for 'X' units of time. The summary of all the requests is submitted to the SEND_GSS_REQUEST to find the best possible QID that can be supported from this GSS component. This GSS.Reg is forwarded to the next component if the current position is a GSS agent or a relay, or else if the GSS component is the transmitter, it responds back to the GSS.Reg with a CGSS.Reg in the forward data path.

```
SEND_GSS_REQUEST (GSS.Dynamic.Request)
        if( GSS.Dynamic.Request < Local.Static.Request)
            if( Local.Dynamic.Provide < GSS.Dynamic.Request)
                if CHECK( GSS.Dynamic.Request, Local status) == OK
                        Local.Dynamic.Request = GSS.Dynamic.Request
                else
                        GSS.Dynamic.Request = BEST PROVIDED
                        Local.Dynamic.Request = BEST PROVIDED
            else
                Local.Dynamic.Request = GSS.Dynamic.Request
        else
            GSS.Dynamic.Request = BEST PROVIDED
```

In the above algorithm, we note that every GSS component keeps three different measures of GSS, a GSS.Static.Request, a GSS.Dynamic.Request, and a GSS.Dynamic.Provide as defined in Section 1.1 (in the registration phase). The

resources promised for this connection is kept on hold till a CGSS.Reg is received. The GSS.Dynamic.Request is used to decide the allocation of the resources for the connections going through this local node.

When receiving a CGSS.Reg, in response to the previous GSS.Reg, the GSS component will allocate the dynamic reservation specified in the CGSS.Reg packet and copy this value into the local GSS component.

The QoS provision algorithm component contains the algorithms required to schedule the data of a connection over the network interfaces. It checks the GSS translation table to arrive at the scheduling parameters to a connection. It will update the transfer statistics of a connection in the GSS table, which can be used by the resource manager and GSS components to allocate resources to a new connection and to the existing connection. The resource manager is a native QoS architecture resource manager, which will access the table through a translator. The resource manager also responds to the GSS component through a standard interface. The specification of the QoS provision component and the resource manager is out of the scope of this chapter.

3 Evaluation of QUANTA

Evaluation of QUANTA is divided into end-to-end QoS provision evaluation, and architectural evaluation. In the first evaluation with the help of Kiviat diagrams, we identify if the application is getting the requested QoS. In the architectural evaluation, we address the evaluation of each of the QoS components. It is also necessary to evaluate the QoS mechanisms for different native QoS architectures. But, this is out of the scope of this chapter, as we are providing the end-to-end

architectural solution.

3.1 End-to-end QoS provision evaluation

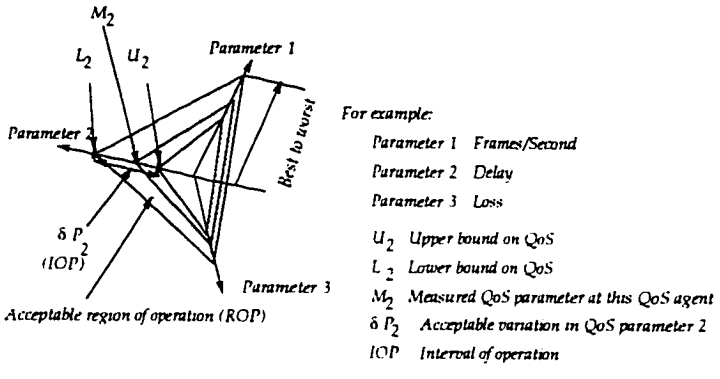


Figure IV.6: Kiviat diagram of QoS measurements

We use the following method to evaluate if an application is obtaining the requested QID. Let (L_i, U_i, W_i) be the end-to-end lower, upper bounds and weight (respectively) of the i^{th} QoS parameter as shown in Figure IV.6. At every e^{th} GSS component we measure the provided QoS parameter (M_{ei}), determine the deviation d_{ei} from the requested QoS, and calculate Figure Of Merit (FOM_{ei}) as shown below against the (L_{ei}, U_{ei}, W_i) assigned to the e^{th} GSS component.

$$\forall\ e\ \in \{\ \textit{End-to-end GSS components}\ \}$$

$$\forall\ i\ \in \{\ \textit{QoS Parameters}\ \}$$

$$d_{ei} = \begin{cases} M_{ei} - L_{ei} & \text{if } M_{ei} < L_{ei} \\ 0 & \text{if } L_{ei} \leq M_{ei} \leq U_{ei} \\ M_{ei} - U_{ei} & \text{if } U_{ei} < M_{ei} \end{cases}$$

$$\text{if } \{\ M_{ei} < L_{ei}\ \} \text{ or } \{\ M_{ei} > U_{ei}\ \}$$

$$FOM_{ei} = \mid d_{ei} * W_i \mid$$

$$FOM_{e,sum} = FOM_{e,sum} + FOM_{ei}$$

$$FOM_i = f\ (FOM_i,\ FOM_{ei})$$

$$FOM_{sum} = FOM_i + FOM_{e,sum}$$

The sum of the FOM_{ei}s is calculated in $FOM_{e,sum}$ for every GSS component, and they are accumulated in the end-to-end FOM_i and FOM_{sum}. The $f()$ operator is dependent on the i^{th} QoS parameter. For example, if i represents loss or delay, then this is a simple addition operation; if i represents throughput then it will represent a minimum operation. At the transmitting end-system, we will know the sum of the obtained QID in FOM parameters, which will be compared with the provided QID by the transmitter. The FOM_i and the FOM_{sum} are calculated for every GSS.Reg packet interval and are used to compute different statistics.

3.2 Architectural evaluation

We devised a three phase evaluation methodology to perform the architectural evaluation of QUANTA. We run a set of base-line experiments with a mix of applications with varied QoS-requirements on a test-bed without QUANTA and use these results to compare the outcome of different phases of QUANTA.

In the first phase of the implementation, as shown in Figure IV.7, we are demonstrating "maintenance of QoS of a connection in the face of competing requests on the end-system." This phase completes the base-line requirements for QUANTA such as providing a QoS communication methodology, QoS translation methodology, QoS allocation and QoS negotiation. We are evaluating this phase using multiple delay-sensitive (video) and throughput-sensitive (data-transfer) connections going through the test-node and disturbing these connections with other traffic (ftp) on the test-node.

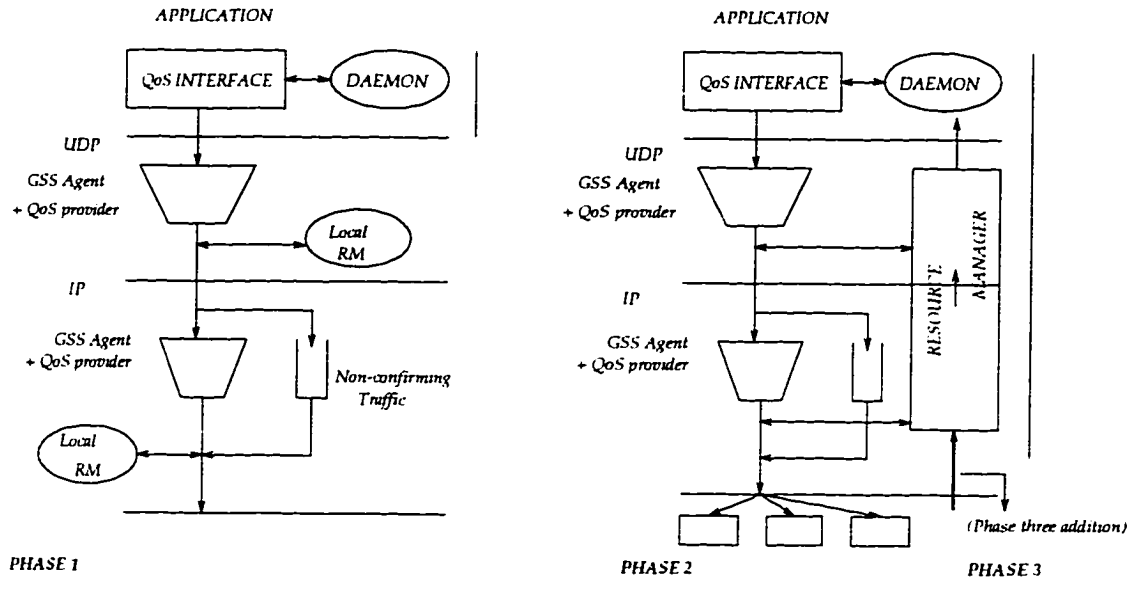


Figure IV.7: Three phases of QUANTA's evaluation

The second phase will provide a "reasonable set of resource choices to enable maximum diversification in QoS requirements." In this phase we will provide the feedback across the end-system protocol layers, multi-service support, and QoS manager. We will evaluate the resource sharing capabilities among multiple classes in this phase by multiplexing applications with varying QoS requirements. For example, through IP we multiplex an audio stream (loss-sensitive) and a video stream (which can accommodate bounded losses) as delay-sensitive applications and data-transfer applications as throughput-sensitive applications.

In the third phase, we will demonstrate "the QoS adjustments upon changes in the host and network conditions." Here we will demonstrate the feedback from the network and use it to dynamically alter the QoS requirements of the applications. We will evaluate this phase by reducing the resources on the host and on the network (through interfering traffic) and verify if the application stays within the requested ROP.

For each of these phases, we will use the figure of merit to compare QUANTA's

performance against the baseline case of running the same applications over currently existing, standard protocol suites. Although the figure of merit will allow us to provide a simple, single number, we shall provide the detailed statistics as well to probe the reasons of better(or lack of) performance by QUANTA.

4 Summary

In this chapter we provided an application-oriented approach to design an end-to-end QoS architecture (QUANTA). We identified the potential QoS issues in such an architecture and proposed a solution to these issues. We proposed ripple-through classification mechanism to classify connections in an application and introduced a Generic Soft State (GSS) and current GSS (CGSS) concepts to accommodate group management of applications and provide dynamic renegotiation of QID. We identified different implementation components to complete QUANTA. These components include application-level components such as a TLI-like QoS interface to the applications, a resource management daemon to maintain and manage the local host system resources; and the protocol-level components include a GSS component, a resource management component, and a QoS provision component. We provided a design of these components and provided an evaluation mechanism to evaluate the end-to-end, user-level QoS provision and a three phase architectural evaluation of QUANTA.

CHAPTER V

RESULTS AND ANALYSIS

Determination: Let us, then, be up and doing

With a heart for any fate:

Still achieving, still pursuing,

Learn to labor and to wait.

— Henry Wadsworth Longfellow

This chapter presents the experiments conducted on the QUANTA test-bed to evaluate the concept we have proposed in this thesis. We use two applications to test QUANTA, the first application being the *modttcp* program (which has been used in the preliminary test chapter) and the second application is the video collaborative application.

The testbed used for these experiments is shown in Figure V.1. This testbed contains two host machines which act as the primary QoS measuring stations and two hosts acting as the load generating machines. As the components developed to demonstrate QUANTA are user-level components, the experiments are independent of the underlying backbone technology. The testbed we use to demonstrate the

concepts of QUANTA is a 10Mb/s Ethernet. We are only interested in demonstrating QUANTA's conceptual correctness rather than providing extensive experimental proof, and it is easy to demonstrate the behavior of QUANTA on a low-bandwidth backbone networks such as Ethernet.

In the next section, we present different sets of experiments conducted to evaluate our work and the rationale behind them.

1 Outline of the experiments

We classify our experiments under two categories. They are:

- Interface overhead experiments: In this set of experiments, we estimate the overhead presented by QUANTA on a communication intensive-application.
- QUANTA concept test experiments: With the help of these experiments we intend to prove the concepts proposed by QUANTA.

In the following sections, we consider each of these set of experiments, elaborate on the experimental procedure, the rationale behind the experiment, the measurements performed and the performance metrics used to evaluate the experimental outcome.

1.1 Interface overhead experiments

These initial sets of experiments measure the overhead incorporated by QUANTA. These experiments are grouped into:

- connection establishment time overhead
- per packet processing overhead and

- per packet data overhead

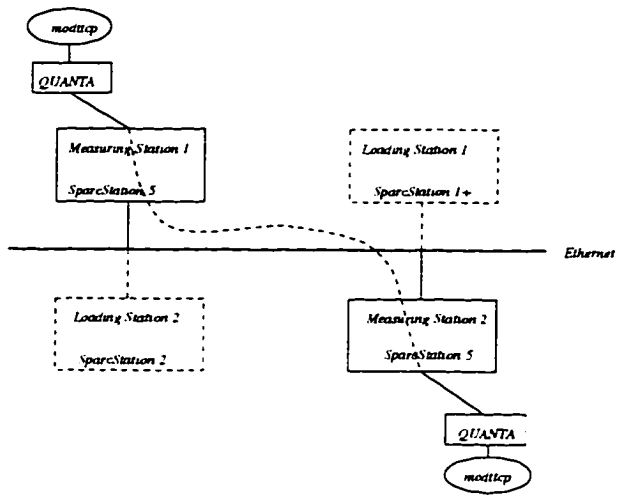


Figure V.1: Testbed used to measure QUANTA overhead

Figure V.1 presents the details of the testbed with respect to the experiments conducted to measure overhead induced by QUANTA. Since these experiments are related to the QUANTA interface to the application, we assume a fixed testbed for all these experiments. The testbed for this set of experiments is an Ethernet backbone with two SUNSparc 5 workstations as the end-system host machines.

Connection establishment time overhead

A communication user application (such as *modttcp* or video collaborative application) using QUANTA will have to perform certain additional tasks before starting to transmit data. These additional tasks include registering the application and the connections with the resource reservation daemon and negotiating the QoS end-to-end with the GSS agents. The processing of these tasks reflects in the performance of the application in terms of an increase in the connection establishment

time. For short-lifetime applications, such as a global timer accessing application*, QUANTA poses an unacceptable overhead. In a moderate-lifetime applications, such as an ftp application, this overhead is debatable. Whereas for a long-lifetime applications, such as a pre-scheduled video collaborative application, QUANTA can provide predictable performance throughout the lifetime of the application without unduly influencing the performance of the application. In the following experiment, we develop a metric which can be used by an application to decide on whether or not to use QUANTA.

We propose that QUANTA can be beneficial to the application if:

$$CE_{b\_factor} = \frac{E[T_{life\_time}]}{E[TC_{with} - TC_{without}]} \geq 10^{\dagger}$$

where

T_{life_time} is the active lifetime of the application.

$E[...]$ is the expected value of the quantity inside the braces.

TC_{with} is the time taken to establish a connection with QUANTA.

$TC_{without}$ is the time taken to establish a connection without QUANTA.

CE_{b_factor} is the QUANTA connection establishment benefit factor.

Table V.1 presents the measurements made on *modttcp* application, which behave as the typical applications mentioned in the table. The values presented in the table are averaged over five runs.

*An application using which a host periodically synchronize its system clock with the universal clock.

†We use the Raj Jain's *rule of thumb* factor [58] as a measure to identify the overhead of QUANTA.

Application Number	Application name	Action Performed	$E[T_{life\_time}]$	$E[TC_{with} - TC_{without}]$	CE_{b_factor}
1	Clock Application	sent a query (20byte) in 256 byte packet	0.06 sec	0.001 Sec	60
2	ftp Application	sent data (100Kbytes) in 1 Kbytes packets	0.11 sec	0.001 Sec	110
3	ftp Application	sent data (1Mbyte) in 2 Kbytes packets	1.2 sec	0.001 Sec	1200
4	Collaborative Application	sent data (50 Mbytes) in 8Kbytes packets	15.9 Mt-s	0.001 Sec	954000

Table V.1: Connection establishment overhead detection experiments

From Table V.1 it can be seen that in application 1 and 2 the benefit factor is marginal, and in applications 3 and 4 the benefit factor is considerable.

Per packet processing overhead

In QUANTA, during the data transfer phase, the applications will exchange GSS packets to monitor the application and react accordingly. In the current implementation of QUANTA on a Solaris OS, this processing is moved form the main application to different GSS threads. Hence the GSS processing overhead and GSS communication processing will not impede the application QoS per se. However, since the GSS packets are sent in-band with the regular data, the application will encounter some per-packet processing overhead. This overhead, similar to the connection establishment overhead, will decrease exponentially as the lifetime of the application increases.

We use the same overhead measuring mechanism, presented in the previous section, in determining the QUANTA per packet processing benefit factor, as follows:

$$PPP_{b\_factor} = \frac{E[T_{life\_time}]}{E[TP_{with} - TP_{without}]} \geq 10$$

where

T_{life_time} is the active lifetime of the application.

$E[...]$ is the expected value of the quantity inside the braces.

TP_{with} is the time taken to transfer data with QUANTA, not including the connection establishment phase.

$TP_{without}$ is the time taken to transfer data without QUANTA, not including the connection establishment phase.

PPP_{b_factor} is the QUANTA per packet processing benefit factor.

In Table V.2, we present the measurements made on the *modttcp* application, which is used to emulate typical applications. The values presented in the table are averaged over five runs.

From Table V.2, we observe that in all the applications the benefit factor is acceptable. This is because in short-lifetime applications (such as application 1) the GSS interval will be higher than the lifetime of the application. Hence, the per-packet processing overhead is negligible. In application 2 the benefit factor is marginal because we accommodated two GSS packet transactions in its lifetime. In applications 3 and 4 the benefit factor is high throughout because of the longer lifetime of the applications.

Per packet data overhead

The data sent by the application using QUANTA will be encapsulated by QUANTA's headers. The packet size, negotiated before the data transfer phase, is fixed and there might be bytes added to the application data packet to obtain this fixed size end-to-end QUANTA packet. Apart from this, without the knowledge of the application, we include the initial exchange of the GSS.init packets to agree to an end-to-end QoS, and the exchange of the GSS.reg packets at regular intervals to monitor the end-to-end QoS and to modify the application QoS[‡]. In the following set of experiments we measure this overhead.

[‡]Note that these overhead measurements will not include the underlying communication protocol (such as TCP/IP) overhead. Also in these experiments we will not consider the exchange of data across the QUANTA and the resource reservation daemon, because it will not add to the traffic on the network.

Application Number	Application name	Action Performed	$E[T_{life\_time}]$	$E[TP_{with} - TP_{without}]$	PPP_{b_factor}
1	Clock Application	sent a query (20 bytes) in 256 byte packet	0.06 sec	0.0011 Sec	54.55
2	ftp Application	sent data (100 Kbyte) in 1 Kbytes packets	0.11 sec	0.01 Sec	11
3	ftp Application	sent data (1 Mbyte) in 2 Kbytes packets	1.2 sec	0.04 Sec	30
4	Collaborative Application	sent data (50 Mbytes) in 8Kbytes packets	15.9 Mt-s	3.25 Sec	293.54

Table V.2: Per packet processing overhead experiments

The per packet data overhead benefit factor is defined as follows:

$$PPD_{b\_factor} = \frac{E[D_{life\_time}]}{E[OH_{static}+OH_{dynamic}]} \geq 10$$

where

D_{life_time} is application data submitted to QUANTA during the active lifetime of the application.

$E[...]$ is the expected value of the quantity inside the braces.

OH_{static} is the static per packet overhead incorporated in to the application data stream. This overhead include GSS.init exchange overhead.

$OH_{dynamic}$ is the dynamic per packet overhead incorporated in to the application data stream. This measurement includes the per packet QUANTA header overhead, per packet extra data padding overhead, and the GSS.reg data packets overhead.

PPD_{b_factor} is the QUANTA per packet data benefit factor.

In Table V.3, we present the measurements made on the *modttcp* application, which is used to mimic typical applications as mentioned in the table. The values presented in the table are averaged over five runs.

From Table V.3, we observe that the benefit factor is low for short-lifetime applications. For example, application 1, is reasonable for medium-lifetime applications and is high for the long-lifetime applications.

Analysis of the overhead benefit factors

Applications with different QoS requirements view the above three overheads in a different perspective. By assigning importance to these overheads, we can identify

Application number	Application name	Action Performed	$E[D_{life\_time}]$	$E[OH_{static} + OH_{dynamic}]$	PPD_{b_factor}
1	Clock Application	sent a query in 256 byte packet	20 bytes	748	0.026
2	ftp Application	sent data in 1 Kbytes packets	100 Kbytes	3129	31.96
3	ftp Application	sent data in 2 Kbytes packets	1 Mbyte	5046	198.17
4	Collaborative Application	sent data in 8Kbytes packets	50 Mbytes	106816	468.09

Table V.3: Per packet data overhead measuring experiments

the overall overhead impact of QUANTA on an application.

Let $< I_{CE}, I_{PPP}, I_{PPD} >$ be the application-dependent importance factor vector. Note that the values given to these parameters are arbitrary and the purpose of these values is only to obtain a quantitative view of the overhead posed by QUANTA.

We assign <0.5, 0.25, 0.25> as the importance factor for the application 1. The rationale behind selecting these fractions is that, in a short-lifetime applications, the connection establishment time is more important than the other two parameters.

For the medium-lifetime applications such as application 2 and 3, we assign <0.25, 0.5 , 0.25> importance factor vector. This is because, in such applications, the PPP overhead is more prominent compared to the other two overheads.

For the long-lifetime application such as application 4, we assign <0.1, 0.5, 0.4> as the importance factor vector. This is because in such applications the amount of data sent on the network and the per-packet processing will contribute more to the QoS degradation of the application.

Let the effective benefit factor to an application be calculated by

$$T_{b\_factor} = I_{CE} \times CE_{b\_factor} + I_{PPP} \times PPP_{b\_factor} + I_{PPD} \times PPD_{b\_factor}$$

Hence, for

application 1 $T_{b\_factor} = 43.644$.

application 2 $T_{b\_factor} = 40.99$.

application 3 T_{b_factor} = 364.543.

application 4 T_{b_factor} = 95734.006.

From the above calculation, we can conclude that the total overhead imposed by QUANTA on different applications is reasonable for short and medium-lifetime applications and is negligible in long-lifetime applications.

1.2 QUANTA concept test experiments

In this set of experiments, we observe and attempt to measure the advantage of using QUANTA. The testbed used in all these experiments is given in Figure V.2. These experiments are grouped into

Sharing resources across applications (multiplexing of applications).

Curbing resources for new applications (a simple admission control).

Preventing application/connection from exceeding the resource allocation (monitoring).

Deviation from predictable performance (provision tests).

In all the following experiments, we use the following set of default parameters globally.

A token is a representative of 256 bytes of data. That is, if an application has 5 tokens at hand, it can send 5×256 = 1280 bytes of data, without being flow-controlled by QUANTA.

A host machine such as M1 in Figure V.2 can supply up to 31250 tokens per second. This number is computed by considering the bandwidth of the

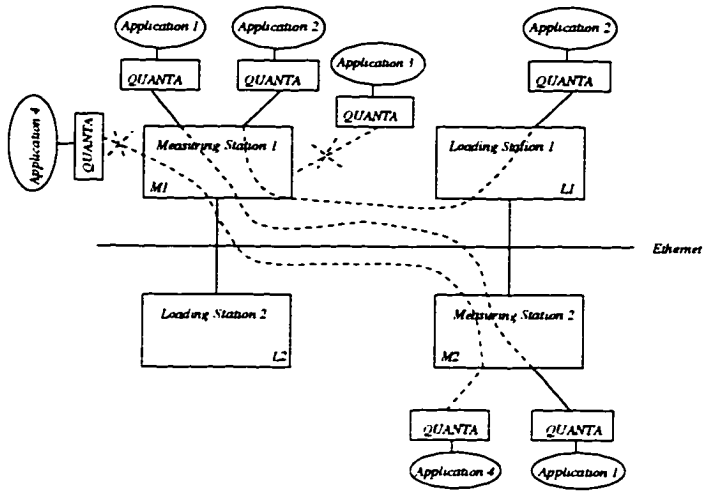


Figure V.2: Testbed used to run the QUANTA conceptual experiments

underlying communication medium. In our case this communication medium is Ethernet. We allocated only 80% of the Ethernet capacity to avoid running into heavy load conditions on the medium.

It is important to observe that the tokens on an end-system are shared by both the incoming and the outgoing applications. We use UDP/IP as the end-system communication protocol, because it is a more predictable protocol compared to TCP/IP and also poses less overhead to the application stream. By using UDP/IP, we can observe the behavior of QUANTA easily. Also note that FOM calculated in this section represents only the receiving side as we do not have the GSS relays or GSS agents in the communication protocols in the current implementation of QUANTA.

Applications multiplexing, admission control, monitoring tests

By using three *modttcp* applications with different QID requests, we check the application notification, termination and resource reservation for these applications in the resource reservation daemon during the connection establishment phase. This would test the multiplexing capability of QUANTA across many applications. During the admission control phase, we attempt to overload the host M1 with more QID requests than the amount of available resources on M1. This will test the admission control of QUANTA during the connection establishment phase. We also monitor the behavior of the applications during the data transfer phase to measure FOM. For all these three experiments, we use three applications with the following characteristics. The weights given to different QoS parameters for the applications are 0.5 for throughput, 0.45 for loss, 0.05 for delay, and zero for the other QID parameters.

Application 1: This application requests for (4.5 Mb/s, 1 Mb/s) of maximum and minimum bounds on the throughput with (0 Mb/s, 0.5 Mb/s) of maximum and minimum bounds on the loss.

Application 2: This application requests for (3.5 Mb/s, 1 Mb/s) of maximum and minimum bounds on the throughput with (0 Mb/s, 0.5 Mb/s) of maximum and minimum bounds on the loss

Application 3: This application requests for (1 Mb/s, 0.5 Mb/s) of maximum and minimum bounds on the throughput with (0 Mb/s, 0.2 Mb/s) of maximum and minimum bounds on the loss.

These application requests are sent to the resource manager in the order of application 1, 2, and 3. We observed that the first two application are accommodated by QUANTA, whereas the last application, application 3, is rejected because

of lack of tokens in the possession of the resource reservation daemon. This test has confirmed the multiplexing capabilities and the admission control of QUANTA during the connection establishment phase. Also by invoking an application, Application 4 from the host M2 to the host M1, we identified that this request is rejected, because the resources (tokens) on M1 are shared by both the incoming and the outgoing applications.

The throughput-time and the loss-time graphs are flat and are less informative in this set of experiments. Hence we present only the FOM calculated on these data. The FOM in case of application 1 and 2 is 0.0124 and 0.0091 respectively. Though, we have expected 0, because of the inaccuracy of the timer routine, QUANTA will sometimes queue data which could lead to buffer overflow. We also observed that the losses due to UDP/IP are nearly zero during these experiments because the applications are rate-controlled in QUANTA and also they do not exceed the host protocol data processing capability (which is approximately 70 Mb/s for a SUNSparc 5).

By increasing the data rate to 5 Mb/s in application 1, we notice that its FOM has increased to 0.37 due to the increase in delay in QUANTA. The FOM of application 2 is still comparable to its FOM in the previous experiment. This experiment shows the penalty an application faces by changing its characteristics.

In conclusion, QUANTA can provide predictable performance under no-load conditions as long as the user application is within its requested bounds. If the application deviates from these bounds, it is penalized by changing the output traffic pattern of the application depending on the weights provided by the application.

QID control tests

In this chapter we present the QoS control tests on the *modttcp* and *VC* applications. We conduct these experiments under no-load, host-load and network-load conditions. The three applications used in these tests have the following characteristics:

modttcp: This application requests for (4.5 Mb/s, 1 Mb/s) of maximum and minimum bounds on the throughput with (0 Mb/s, 0.5 Mb/s) of maximum and minimum bounds on the loss. The weights given to different QoS parameters for this application are 0.5 for throughput, 0.45 for loss, 0.05 for delay, and zero for the other QID parameters.

VC: A Video Collaborative (*VC*) application is a delay-sensitive application with bounds on losses. The throughput request of this application is 3.5 Mb/s upper bound and 1 Mb/s lower bound. The delay bound on this application is 0 msec upper bound and 10 msec lower bound, and the loss bounds are 0 Mb/s to 1 Mb/s. The weights given to different QoS parameters for the application are 0.5 for delay, 0.25 for loss, 0.25 for throughput and zero for the other QID parameters.

ftp: An emulated file transfer application (ftp) is used to generate background load on the Ethernet. This application is rate-controlled to send data at different data rates on the Ethernet.

In Figures V.3 and V.4, we present a case study of the experiments conducted under the network-load, the host-load, and the no-load conditions. Under no-load condition, the host and the networks are not loaded. During the host-load conditions, the host machine is loaded with a 40% CPU intensive job and an 1Mb/s ftp application is used as an incoming data transfer application. In the network load

condition, an ftp application is used to send data across the Ethernet between L1 and L2.

These graphs contain the network-load, host-load and no-load experiments in that order. The first dip in the throughput graph is due to the network-load condition. We can observe that during this transition the application throughput recovers to a stable state less than the maximum negotiated throughput. This is because of the back-off algorithm we have implemented in the QUANTA application library. Also observe that the delay is also bounded because of the reaction to the changing state on the network. The loss graph shows that the *VC* application encounters more loss than the loss characteristics of the *modttcp* application. The similar characteristics can also be observed from the host-load conditions.

The application's behavior under different network conditions are presented in Table V.4. It can be seen from the table and the case study that the QID of different application characteristics are maintained by QUANTA. Here again we assume that as long as the application maintains FOM less than 1 throughout the application data transfer phase, we assume that it is meeting tits QoS bounds.

We observed FOM of a *VC* and a *modttcp* application by changing the load on the network using a *modttcp* application running between L1 and L2. We collected the throughput, delay, and jitter measurements as the time progress under different network load conditions. Under low network load conditions such as 10% and 20% as shown in Table V.4, the FOM of both the applications are within their specified QID bounds (as shown in Figure V.3, V.4 for 10% network load conditions). As the load on the network is increased, in a *modttcp* application, the throughput glitches caused increase in FOM, in a *VC* application the change in delay characteristics and the increase in loss caused the increase in FOM. By making QUANTA sensitive

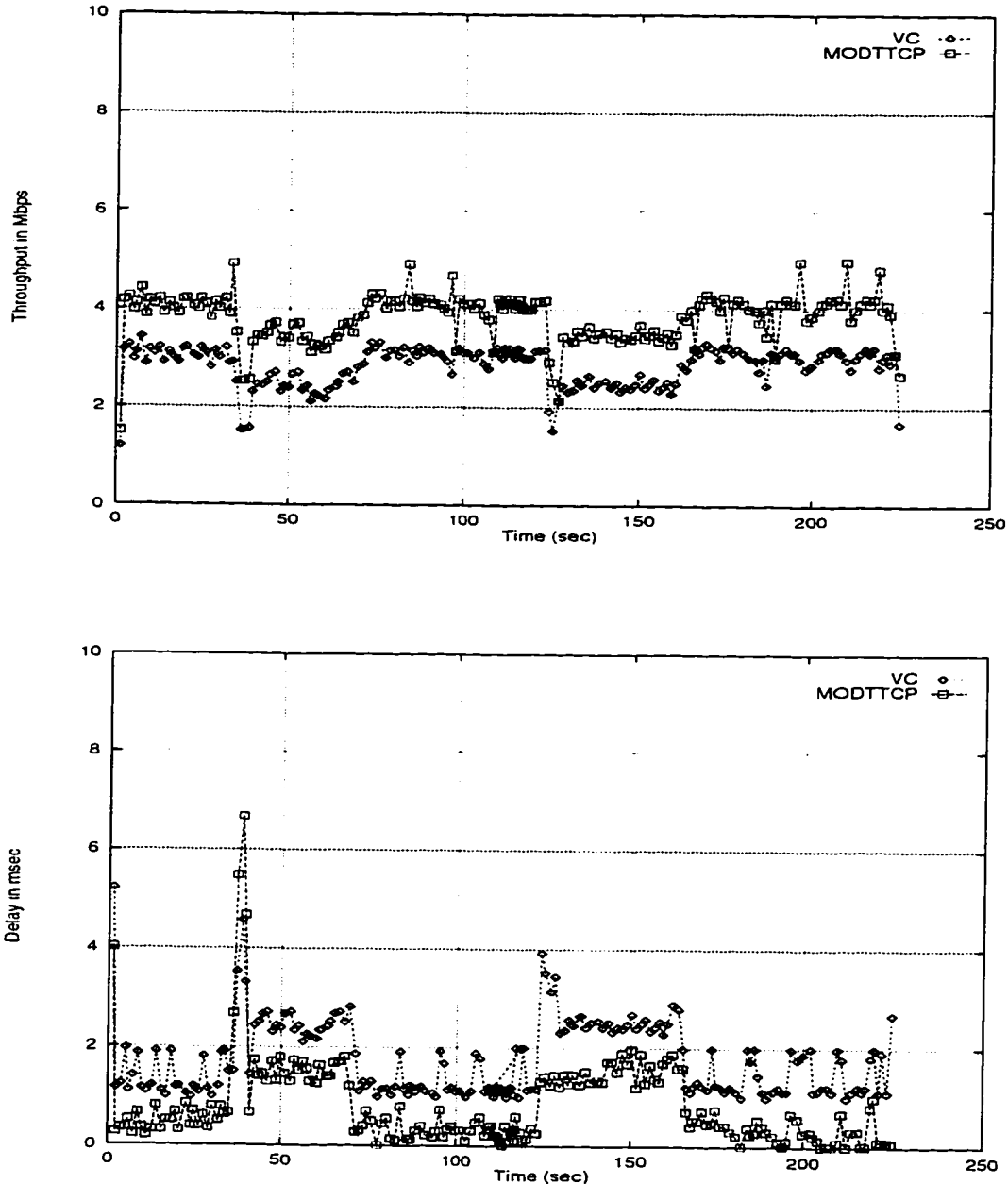


Figure V.3: Throughput and delay graphs for the VC and the MODTTCP applications under 10% network load condition, 10% host load condition, and under no-load condition

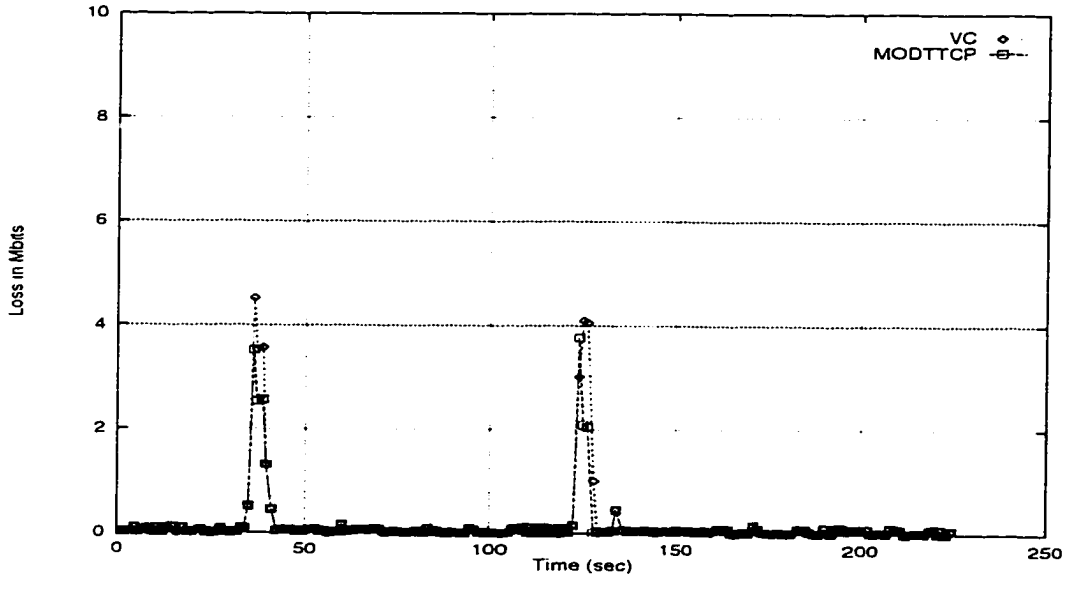


Figure V.4: Loss graphs for the VC and the MODTTCP applications under 10% network load condition 10% host load condition, and under no-load condition.

Network Load	Application Number	Receiver FOM_{thput}	Receiver FOM_{delay}	Receiver FOM_{loss}	Receiver FOM
10%	*VC*	0	0	0	0
10%	*modttcp*	0	0	0	0
20%	*VC*	0.025	0	0.05	0.075
20%	*modttcp*	0.025	0.05	0	0.075
30%	*VC*	0.1	0	0.15	0.25
30%	*modttcp*	0.15	0.11	0	0.26
40%	*VC*	0.28	0.1	0.19	0.57
40%	*modttcp*	0.21	0.31	0	0.52

Table V.4: FOM measurements for the *VC* and *modttcp* applications at different network load conditions

to jitter measurements rather than throughput, we predict that tighter control on FOM can be achieved.

The penalty paid to keep the applications within its QID bounds are the applications itself, because of the current mechanisms included in QUANTA are reactive rather than preventive. To make these mechanisms preventive, it is necessary to modify the scheduling algorithms in the communication protocols as discussed in the issues chapter (Chapter 3) and to prompt the standards committees to incorporate these changes in currently not in the scope of the author. But the prediction of the author is that by making the changes in the protocol suites such as IP and ATM (as it is in progress) and by using QUANTA, it is the author's prediction that the achievement of predictable performance is a reachable goal.

2 Summary

In this chapter, we have analyzed the overhead imposed by QUANTA and the advantages of QUANTA under different load conditions. We observed that the QUANTA overhead is minimal for the long-life time applications. For short and medium-lifetime applications, the overhead is reasonably small. In terms of benefit, the experiments in this chapter show that, transparent to the user, QUANTA will automatically adjust the QoS parameters and weights for an application to meet the changing network and host environment.

Specifically, by comparing the graphs in the preliminary results chapter in sections 2.3 and 2.4 with the results in section 5.1.2, we can observe that the application is within the QoS bounds throughout the data transfer phase although it is penalized according to the weights specified by it. In *VC* and *modttcp* applications, the throughput and the delay characteristics are altered to accommodate the load

on the network. In Figure II.10 without QUANTA, the throughput of a connection degrades almost to 1 Mb/s though the available capacity on the physical medium is nearly 40 Mb/s. The delay characteristics of a connection can suffer (the maximum delay observed is 350 msec), as shown in Figure II.10, without QUANTA. In contrast, Figure V.3 shows that the delay characteristics are within the user specified bounds when using QUANTA. The maximum delay observed by using QUANTA is less than 10 msec. The reaction time to a change in the status of the load condition is nearly two GSS time periods, which in our case is 40 msec. We can make QUANTA more sensitive to load changes by reducing the GSS/CGSS period, but this increases the overhead by QUANTA as the GSS/CGSS packets are sent in-band with the application data. The changes in the throughput are the result of the monitoring mechanism using the GSS/CGSS packets, which flow between the participating applications. Under high network load conditions, during the transition between two stable throughputs, the application is crossing the requested QID boundaries. This can further be controlled by making QUANTA more sensitive to jitter bounds rather than depending only on the throughput measurements. We can also observe that as discussed earlier the effective throughput of the medium can also be increased by reacting to the changing conditions of the network. This can be demonstrated by considering the aggregate throughput as shown in Figure II.10 (which is without QUANTA and by considering the same in Figure V.3). We have noticed near 80% utilization of the channel with QUANTA, whereas the same without QUANTA under loaded conditions is only 40%.

CHAPTER VI

CONCLUSIONS AND OPEN PROBLEMS

Facts: I often wish ... that I could rid the world of the tyranny of facts. What are facts but compromises? A fact merely marks the point where we have agreed to let investigation cease.

— Bliss Carman

In this thesis we provided an application-oriented approach to designing an end-to-end QoS architecture (QUANTA). We identified the potential QoS issues in such an architecture and proposed a solution to these issues. We proposed a ripple-through classification mechanism to classify connections in an application and introduced Generic Soft State (GSS) and current GSS (CGSS) concepts to accommodate group management of applications and provide dynamic re-negotiation of QID. To realize these concepts we have designed and proto-typed components such as a TLI-like QoS interface to the applications, a resource management daemon to maintain and manage the local host system resources, and at the protocol-level, a GSS component, a resource management component, and a QoS provision component. We developed a mechanism to evaluate the end-to-end, user-level QoS provisions and a three-phase architectural evaluation of QUANTA.

We have analyzed the overhead imposed by QUANTA and the advantages of using QUANTA under different load conditions. We observed that the QUANTA overhead is minimal for the long-life time applications. For short and medium-life time applications the overhead is reasonably small. We noticed the seamless control of QUANTA in negotiating, managing and maintaining QID of different classes of applications under different host and network load conditions. We also observed the effect of weights on the application characteristics.

By comparing the graphs in the preliminary results chapter in sections 2.3 and 2.4 with the results in section 5.1.2, we can observe that the application is within the QoS bounds throughout the data transfer phase although it is penalized according to the weights specified by it. In *VC* and *modttcp* applications, the throughput and the delay characteristics are altered to accommodate the load on the network. It is also observed that it takes two GSS/CGSS time periods to accommodate a change. We can make QUANTA more sensitive to load changes by reducing the GSS/CGSS period, but this increases the overhead by QUANTA as the GSS/CGSS packets are sent in-band with the application data. The changes in the throughput are the result of the monitoring mechanism using the GSS/CGSS packets, which flow between the participating applications. Under high network load conditions, during the transition between two stable throughputs, the application is crossing the requested QID boundaries. This can further be controlled by making QUANTA more sensitive to jitter bounds rather than depending only on the throughput measurements. Another observation that can be made in this chapter is the effective throughput on the network can also be increased by reacting to it. At the same time, we can observe from the UDP throughput versus time graph in Figure II.7 if the application is not controlled, it can change the characteristics of the other

applications using the same network medium.

As described in this thesis, QUANTA still has some limitations which need to be resolved. How do we integrate different application requirements into a QID specification? Currently, we have demonstrated a simple mechanism to integrate common applications into the QUANTA realm. In the specification of QID of an application we have included synchronization and cost as two parameters. How to incorporate such application knowledge into QID negotiations and to the communication and the QoS architecture is an open question. We have provided some hooks to support re-negotiation using QUANTA; we have left open how to integrate this concept into the QUANTA's interface. The weights algorithm, which is used to react to network and host-load condition algorithms, could be improved to react more effectively to the changing state of the application traffic characteristics. The protocol-level components of QUANTA need to be implemented to guarantee the end-to-end application-level QoS guarantee, which includes the scheduling portion in the protocols using QID. The implementation of lateral translation of QID needs support from different protocol-suites and their QoS supporting protocols.

To keep the applications within its QID bounds currently only the applications itself is penalized, because of the mechanisms included in QUANTA are reactive rather than preventive. To make these mechanisms preventive, it is necessary to modify the scheduling algorithms in the communication protocols as discussed in the issues chapter (Chapter III). QUANTA by itself is a QoS interface between the applications and the network protocol suites. Any application could take advantage of QUANTA in registering itself with the network protocols, negotiating it's QID with the network and react to the changing QoS conditions in the network and on the host. However, to guarantee QID and prevent other non-QoS confor-

mant applications from interfering with the QoS-conformant application (such as the applications using QUANTA) it is essential to have a QoS scheduler inside the communication protocol suite. The standards committees for protocol suites such as IP and ATM are investigating into different QoS schedulers. Although QUANTA is developing different QoS schedulers proposed in chapter III, it can work with any other scheduler adopted by the standards committee. Hence, QUANTA in conjunction with the QoS schedulers in the protocols can provide predictable performance to a wide spectrum of applications.

BIBLIOGRAPHY

[1] "IRI Home Page," *http://www.cs.odu.edu/~tele/iri.*

[2] S. Dharanikota, K. Maly, and C. M. Overstreet, "Performance evaluation of TCP(UDP)/IP over ATM networks," tech. rep., Department of Computer Science, Old Dominion University, # TR_94_23, September 1994.

[3] S. Dharanikota, K. Maly, D. E. Keyes, and C. M. Overstreet, "An application-oriented analysis of TCP/IP in high speed LANs," *ATNAC-94*, December 1994. Melbourne.

[4] D. Borman, R. Braden, and V. Jacobson, "TCP Extensions for High Performance," *Request for comments 1323*, May 1992.

[5] J. Postel, "Transmission Control Protocol," *Request for comments 793*, November 1981.

[6] C. Partridge, "Workshop Report: Internet Research Steering Group Workshop on Very-High-Speed Networks," *Request for comments 1152*, April 1990.

[7] B. Leiner, "Critical issues in high bandwidth networking," *Request for comments 1077*, November 1988.

[8] R. Fox, "TCP big window and NAK options," *Request for comments 1106*, June 1989.

[9] A. McKenzie, "Problem with the TCP big window option," *Request for comments 1110*, August 1989.

[10] R. Braden and V. Jacobson, "TCP extensions for long-delay paths," *Request for comments 1072*, October 1988.

[11] D. E. Comer, "TCP Buffering And Performance Over An ATM Network," tech. rep., Purdue, # CSD-TR_94_26, March 1994.

[12] V. Jacobson, "Congestion avoidance and control," *ACM Computer Communication Review, Proceedings of the Sigcomm '88 Symposium*, vol. 18, August 1988. Stanford, CA.

[13] A. Danthine, O. Bonaventure, Y. Baguette, G. Leduc, and L. Leonard, "OSI 95 Enhancements and The New Transport Services," *Local Area Network Interconnection*, pp. 1–22, 1993.

[14] D. Ferrari, "Real-time communication in an internetwork," tech. rep., University of California, Berkeley, # CSD-TR_94_034, 1994.

[15] C. Topolcic, "Experimental Internet Stream Protocol, Version 2 ST-II," *Request for comments 1190*, October 1990.

[16] W. Prue and J. Postel, "Queuing algorithm to provide type-of-service for IP links," *Request for comments 1046*, February 1988.

[17] P. Almquist, "Type of Service in the Internet Protocol Suite," *Request for comments 1349*, July 1992.

[18] W. Fischer *et al.*, "Data Communications Using ATM: Architectures, Protocols, and Resource Management," *IEEE Communications*, vol. 32, August 1994.

[19] T. ATM-Forum, *ATM: User-Network Interface specification - version 3.0.* Prentice Hall Publications limited, 1993.

[20] P. Boyer and D. Transchier, "A reservation principle with application to the ATM traffic control," *Computer Networks and ISDN systems*, vol. 24, 1992.

[21] B. Doshi and H. Heffes, "Overloading performance of an adaptive, buffer-window allocation scheme for a class of high speed networks," *Teletraffic and Data Traffic in a Period of Change, Jensen and Iversen (eds.)*, 1991. Elsevier.

[22] P. Newman, "ATM Local Area Network," *IEEE Communications*, vol. 32, March 1994.

[23] N. Yin and M. G. Hluchyj, "On closed-loop rate control for ATM cell relay networks," *Proc. IEEE INFOCOM*, June 1994. Toronto.

[24] H. T. Kung *et al.*, "Use of link-by-link flow control in maximizing ATM network performance: Simulation results," *Proc. IEEE Hot Interconnection Symposium*, August 1993. Palo Alto, California.

[25] J. Crowcraft, Z. Wang, A. Smith, and J. Adams, "A Rough Comparision of the IETF and ATM Service Models," *IEEE Network*, November/December 1995.

[26] S. Dixit and S. Paul, "MPEG-2 over ATM for Video Dial Tone Networks: Issues and Strategies," *IEEE Network*, September/October 1995.

[27] B. Braden, D. Clark, and S. Shenkar, "Integrated Services in the Internet Architecture: an Overview," *Request for comments 1633*, June 1994.

[28] L. Zhang *et al.*, "Resource ReSerVation Protocol," *IEEE Network*, September/October 1993.

[29] "Tenet Home Page," *http://tenet.berkeley.edu.*

[30] S. Boking, "TIP's performance Quality of Service," *IEEE Communications Magazine*, August 1995.

[31] A. Banerjea *et al.*, "The Tenet Real-Time Protocol Suite: Design, Implementation, and Experiences," tech. rep., International Computer Science Institute, # CSD-TR_94_059, November 1994. Berkeley, CA.

[32] "BISDN access signaling system DSS2 (Digital Subscriber Signaling System No. 2)," *ITU-T Recommendation Q.2931.*

[33] E. S. Domino *et al.*, "Overview of ATM networks: functions and procedures," *Computer communications review*, vol. 14, pp. 615–626, December 1991.

[34] J. Y. L. Boudec, "The Asynchronous Transfer Mode: a tutorial," *Computer Networks and ISDN Systems*, pp. 279–309, 1992.

[35] *ATM forum signaling group discussions*, June 1995.

[36] "STREAMS Programmer's Guide," *SunSoft Technical Manual.* SunOS 5.1.

[37] F. Bonomi and K. W. Fendick, "The Rate-Based Flow Control Framework for the Available Bit Rate ATM Service," *IEEE Network*, vol. 9, March/April 1995.

[38] H. T. Kung and R. Morris, "Credit-Based Flow Control for ATM Networks," *IEEE Network*, vol. 9, March/April 1995.

[39] D. Hong and T. Suda, "Congestion Control and Prevention in ATM Networks," *IEEE Network Magazine*, pp. 10–16, July 1991.

[40] P. Moldeklev, K.and Gunningberg, "Deadlock situations in TCP over ATM," *4th International IFIP workshop on Protocols for HSN*, August 10-12 1994. Vancouver, B.C.Canada.

[41] C. Papadopoulos and G. M. Parulkar, "Experimental evaluation of SUNOS IPC and TCP/IP protocol implementation," *IEEE/ACM Transactions on Networks*, vol. 1, April 1993.

[42] S. Dharanikota, K. Maly, C. M. Overstreet, and R. Mukkamala, "Missing end-system components: A case-study," tech. rep., Department of Computer Science, Old Dominion University, # TR_95_15, June 1995.

[43] S. Dharanikota, K. Maly, and C. M. Overstreet, "Three phases of QUANTA: Design specifications and implementation details," tech. rep., Department of Computer Science, Old Dominion University, http://www.cs.odu.edu/~dhara_s/QUANTA.

[44] S. Dharanikota, K. Maly, and C. M. Overstreet, "Performance evaluation of TCP(UDP)/IP over ATM networks," tech. rep., Department of Computer Science, Old Dominion University, # TR_95_23, September 1995.

[45] A. Campbell, G. Coulson, and D. Hutchison, "A quality of Service Architecture," *Computer communications review*, vol. 24, April 1994.

[46] A. K. J. Parekh, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks," *MIT Laboratory for Information and Decision Systems, LIDS-TH-2089*, February 1992. Cambridge, Mass.

[47] The ATM Forum Technical Committee, "ATM User-Network Interface Specification: Version 3.1," September 1994.

[48] The ATM Forum Technical Committee, "Traffic Management Specification: Version 4.0," February 1996. ATM Forum/95-0013R10, Straw Vote.

[49] A. Banerjea, D. Ferrari, B. A. Mah, M. Moran, D. C. Verma, and H. Zhang, "The Tenet Real-Time Protocol Suite: Design, Implementation, and Experiences," tech. rep., University of California, # CSD-TR_94_059, November 1994. Berkeley, CA.

[50] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," *Request for comments 1883*, December 1995.

[51] C. Partridge, "A Proposed Flow Specification," *Request for comments 1363*, September 1992.

[52] L. Zhang *et al.*, "Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification," *Internet Draft, draft-ietf-rsvp-spec-10.txt*, February 1996.

[53] S. Dharanikota and K. Maly, "QUANTA:Quality of Service Architecture for Native TCP/IP over ATM networks," tech. rep., Department of Computer Science Old Dominion University, #CSD-TR_96_01, February 1996. avaiable at http://www.cs.odu.edu/~dhara_s/QUANTA, also accepted at HPDC Conference.

[54] A. Varma *et al.*, "An efficient rate allocation algorithm for ATM networks providing min-max fairness," *High Performance Networking VI*, September 1995.

[55] S. Dharanikota and K. Maly, "QoS issues in HQNs: QUANTA's approach," tech. rep., Department of Computer Science Old Dominion University. http://www.cs.odu.edu/~dhara_s/QUANTA, also accepted at Eighth IEEE Workshop on Local and metropolitaArea Networks.

[56] "Some details of QUANTA," *http://www.cs.odu.edu/~dhara_s/QUANTA*.

[57] I. Stoica and H. Abdel Wahab, "Earliest Eligible Virtual Deadline First: A Flexible and Accurate Mechanism for Proportional Share Resource Allocation," November 1995. Revised January 26, 1996.

[58] R. Jain, *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc., 1991.

APPENDIX A

STATE DIAGRAMS AND ALGORITHMS

In Figures A.1 and A.2 we present the state diagrams of QUANTA involved in different stages of an application communication. These state diagrams are self-explanatory after understanding the functionality of QUANTA's application interface. In Figure A.3, we present the weights algorithm used by QUANTA to identify the application requirements by deciphering the weights and QTG map of the application. Figure A.4 presents the algorithms used by the send side, and in Figure A.5 the receive side of the QUANTA's library to maintain, monitor and react to the current state of the host and network status.

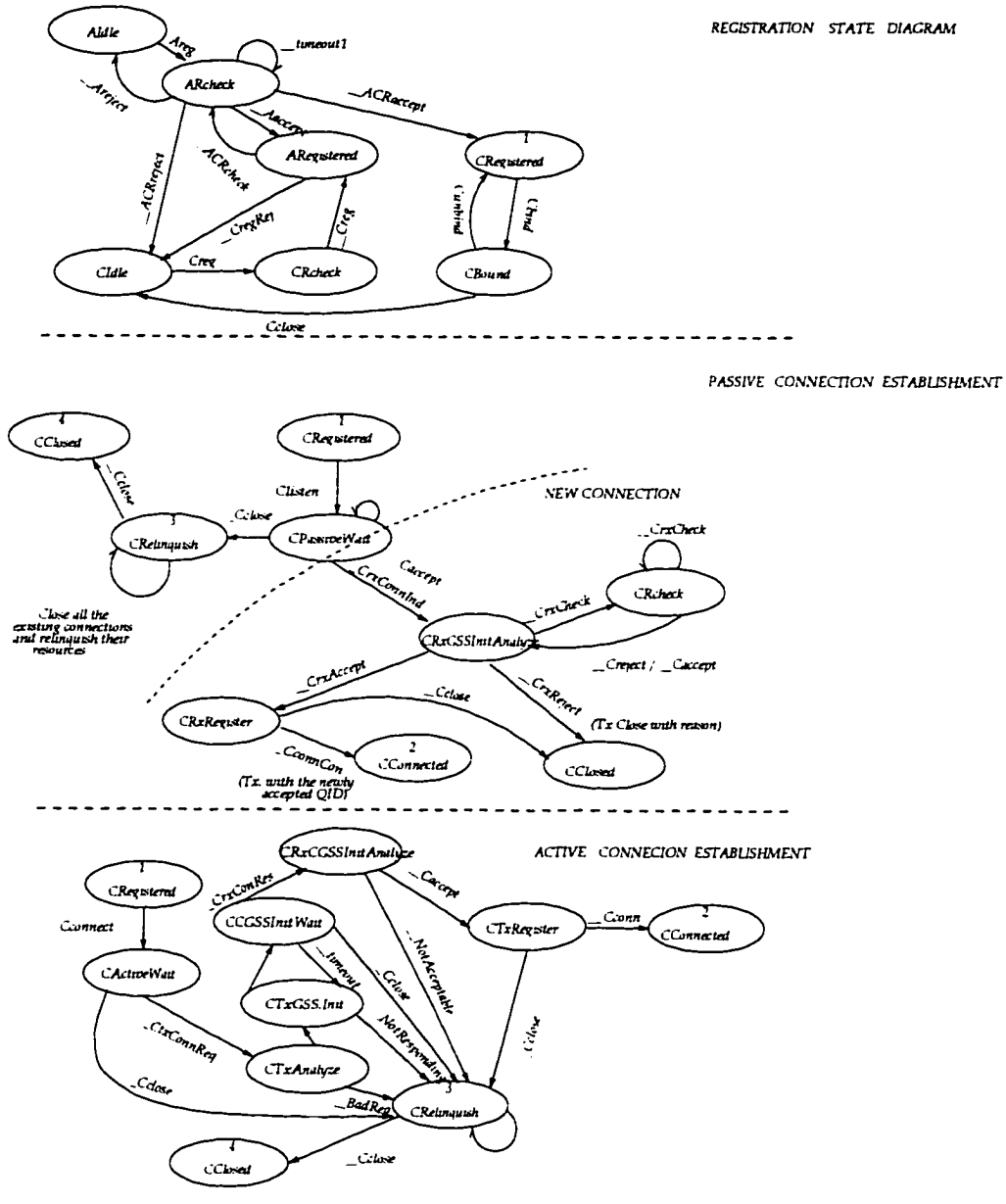


Figure A.1: State diagrams of QUANTA's application library.

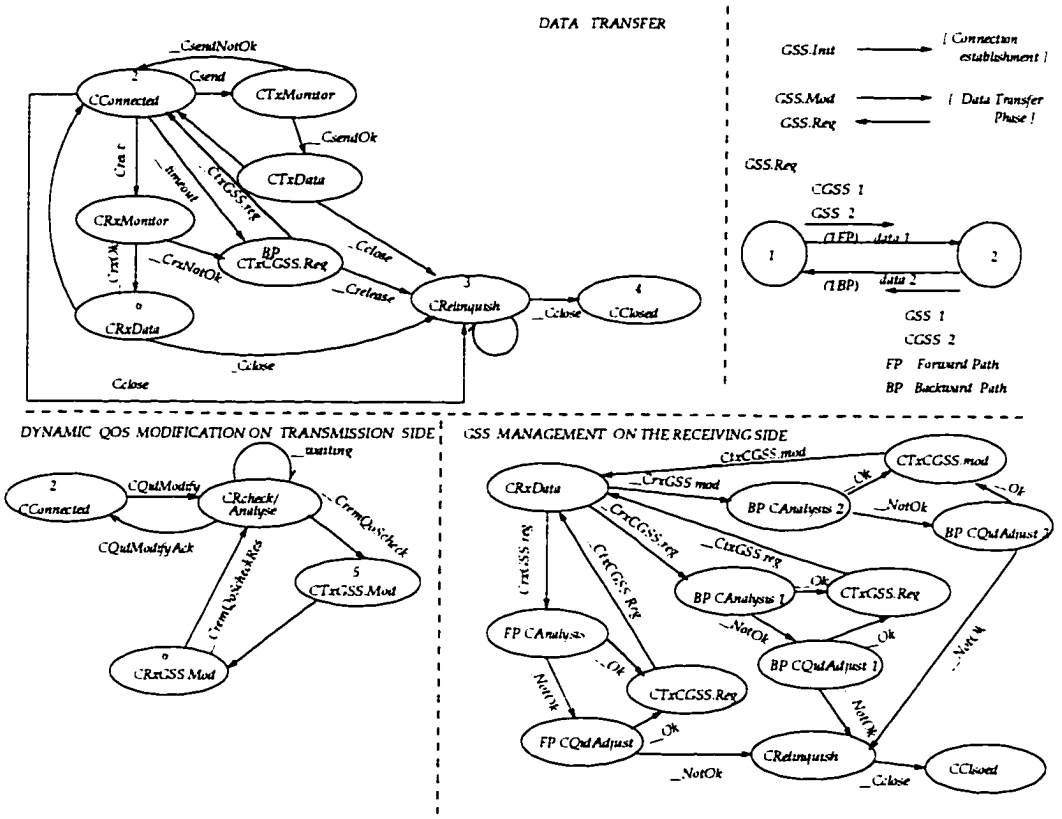


Figure A.2: State diagrams of QUANTA's application library (continued).

```
WEIGHTS ALGORITHM  /* Used to deterimine the "plan of action" for the scheduler in case of congestion */
                   /* This algorithm is run only at the connection establisahment time */

if ( wloss < 0.5 )      /* Meddle with loss - can afford high losses */
    if ( ( QTG && DELAY) == TRUE )      /* Delay Sensitive connection */
        if ( wthput < wloss)            /* bursty losses or loss rate */
            if ( wjitter <= wdelay) /* Can have bursty losses */
                Decision = BURSTY
            else                        /* Only increase the loss ration */
                Decision = LOSS_RATE
        else
            Decision = LOSS_RATE

    else        /* Throughput sensitive connection */       /* Meddle with delay */
        if ( wdelay <= wloss )          /* Can have bursty losses */
            Decision = BURSTY
        else                            /* Only increase the loss rate */
            Decision = LOSS_RATE
        /* NOTE : "wjitter" is not considered because for throughput sensitive connections this parameter does not matter */


else if ( wloss >= 0.5)     /* connctions are loss-sensitive */
    if( ( QTG && DELAY) == TRUE)  /* Delay Sensitive connection */
        if( wloss != 1.0)
            if( wjitter < wdelay)       /* Can have bursty losses */
                Decision = BURSTY
            else                        /* Jitter sensitive */
                Decision = LOSS_RATE
        else                            /* Typical audio connections - what to do !! */
            Decision = LOSS_RATE
    else        /* Throughput sensitive class */
        if( wloss == 1.0)               /* delay */
            Decision = DELAY
        else
            Decision = LOSS_RATE
```

Figure A.3: QUANTA application library's weights algorithm

```
SEND ALGORITHM

  CSend( CCB, PKT )
        PKT.SequenceNumber = ++ CCB.FPSeqNum
        PKT.Time  = GetPresentTime();

        if( CCB.FPremainingTokens > 0 )
              QPKT( FPQ, PKT)
        else
            if( Decision == DELAY )
                  delay( CCB->FPDelay)
            else if (Decision == LOSS_RATE)
                  Free ( PKT);
            else
                  Free ( PKT);

  SendDataThread ( CCB )
     while ( TRUE )
        if ( QStatus ( CCB.FPDataToTx) )
              SendData ( PKT);
        else
              wait till PKT exist in the Q   /* Using broadcast message */

        delay ( ScheduleTime)

  RecvGSS.Reg ( CCB , GSSPkt)

     if( GSS.MinThroughput < GSSPkt.Throughput  < GSS.MaxThroughput )
              SET TOKENS ACCORDING TO THE NEW THROUGHPUT
     else
              Inform the user about the current state of the network
```

Figure A.4: QUANTA application library's send-side algorithm

Receiver measurements:

$$Jitter = |R2 - R1| - |S2 - S1|$$

$$Delay = (R2 - S2) +/- Clock\ Difference$$

$$Loss\ Rate = \frac{Lost\ Sequence\ numbers}{Total\ Sequence\ numbers}$$

$$Throughput = \frac{Number\ of\ bytes\ received}{Time\ between\ CGSS\ packets}$$

These measurements are averaged over CGSS interval which is measured as "N" packet interval.

Congestion condition is identified initially by the Jitter bounds and then by the Loss Rate.

In either case throughput is decreased by increasing the delay or increasing the loss rate (depending on the "weights").

```
PrevPktSentTimeStamp = 0;          PrevPktRcvTimeStamp = 0 ;
PrevPktSeqNum = 1;
MONITOR ( CCB, PKT )
    CCB.TempJitter = CCB.TempJitter + ( CurrentTime - PrevPktRcvTimeStamp) - ( PKT.Time - PrevPktSentTimeStamp)
    PrevPktSentTimeStamp = PKT.Time                PrevRcvTimeStamp  = CurrentTime
    CCB.LostPkts = LostPkts + (PKT.SeqNum - PrevPktSeqNum)
    PrevPktSeqNum = PKT.SeqNum
    /* Delay measurement needs clock difference  and throughput can be calculated from the above information itself */


PrevCCGSclock = 0 ;
CGSS.Reg Receive ( CCB )
    if( ((CCB.TempJitter) / (TotalPkts - CCB.LostPkts)) > 0.1 * CCB.Jmax)  OR  ( CCB.LostPkts/ TotalPkts) > 0.1 * LossRate )
            CGSS.Throughput = PrevCGSS.Throughput * ( 1 / ( 2 *  celing( AvgJitter / 10)))
    else if (  AvgJitter <  0.1 CCB.Jmax ) AND ( LossRate > 0.1 * LossRate_max)
        AND  ( CCB.MaxThroughput - CGSS.Throughput > 0.1 * CCB.MaxThroughput)
            CGSS.Throughput = PrevCGSS.Throughput *  0.1 * CCB.MaxThroughput
```

Figure A.5: QUANTA application library's receive-side algorithm

APPENDIX B

ACRONYMS

AAL	ATM Adaptation Layer
ATM	Asynchronous Transfer Mode
BECN	Backward Error Congestion Notification
BSD	Berkeley Software Division
CAC	Connection Admission Control
CCITT	The International Telegraph and Telephone Consultative Committee
DQDB	Distributed Queue Dual Bus
FDDI	Fiber Distributed Data Interface
FECN	Forward Error Congestion Notification
FPS	Frames Per Second
GN	Gigabit Network
HSN	High Speed Network
HQN	High Quality Network

(Continued in the next page)

IP	Internet Protocol
LAN	Local Area Network
MAN	Metropolitan Area Network
MPEG	Motion Picture Expert Group
MSS	Mean Segment Size
OSI	Open System Interconnection
QoS	Quality of Service
RFC	Request For Comments
RTT	Round Trip Time
RTTM	Round Trip Time Measurement
ST	Stream Protocol
SWS	Silly Window Syndrome
TCP	Transmission Control Protocol
TOS	Type of Service
UDP	User Datagram Protocol
VC	Video Collaborative application

VITA

Sudheer Dharanikota was born in Srikakulam, A.P., India, on June 14^{th}, 1967. He received his Bachelor of Technology in Electronics and Communications Engineering from Nagarjuna University, Vijayawada, India, in August, 1988; and received his Master of Technology in Electrical Communications Engineering from Indian Institute of Science (IISc), Bangalore, India, in January, 1990. He worked as a Scientific Officer at ERNET (Education and Research in Networking), IISc from January, 1990 until August, 1992, when he joined for his PhD at Old Dominion University (ODU). From August, 1992 until August, 1996 he was working on his Ph.D degree in the department of Computer Science at ODU, Virginia, USA.

Permanent address: Department of Computer Science
Old Dominion University
Norfolk, VA 23529
USA

This dissertation was typeset with LaTeX† by the author.

†LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.