MULTIMEDIA REALTIME TRANSPORT PROTOCOL OVER ATM NETWORK

by

ZHENJUN ZHU

A thesis submitted to the

Department of Computing and Information Science

in conformity with the requirements for

the degree of Master of Sciences

Queen's University

Kingston, Ontario, Canada

December 1996

Abstract

As ATM (Asynchronous Transfer Mode) networking switches and access equipment begin to be deployed on a wider scale, the development of applications which utilize high-bandwidth transport services is becoming a focus of research. TCP/IP was not designed to handle the real-time traffic generated by broadband multi-media applications so a broadband transport protocol which supports the development of broadband real-time applications is needed.

We propose a transport protocol middleware which includes broadband-specific transport service functions such as virtual connection setup, bandwidth reservation, and session synchronization, and which provides a development environment that integrates real-time delivery, quality of service guarantee, control and management. This transport service middle-ware is based on RTP (Real-time Transport Protocol) from IETF (Internet Engineering Task Force). The thesis discusses the design and implementation of QRTP (Queen's Real-time Transport Protocol) and evaluates the performance of the software.

Acknowledgments

I would like to thank Dr. Pat Martin, my supervisor and co-investigator in Queen's Multimedia Networking project, for his patience, guidance, suggestions, and generous conference sponsorship. Without these , I could never have completed this thesis.

I would also like to thank Dr. H.T.Mouftah of Department of Electrical and Computer Engineering, my co-supervisor and investigator of Queen's Multimedia Networking project, for his support and constant feedbacks into my work.

Thanks also go to Wendy Powley and other members of Database System Laboratory, for their support and friendship which is a main drive force in my work.

I thank Natural Science and Engineering Research Council of Canada (NSERC) for awarding me a PGS-A scholarship which enables me to carry out this research. I therefore also thank Department of Computer Science, University of Western Ontario from where I applied for this award.

Setting up of Queen's University ATM Multimedia Networking Testbed involved

effort from other people, including Greg Macleod of Electrical and Computer Engineering, Andy Hooper of Information Technology Service, Tom Bradshaw, Gary Powley, and Dave Dove of Computing and Information Science department. I would like to give them my sincere thanks on all the help I have got from them. Newbridge is generous enough to provide us with two ATM switches which make our research practically possible. I also thank heads of both Department of Electrical and Computer Engineering, and Department of Computing and Information Science, for providing important equipment funding used in connecting these switches, under a tight operation budget.

Contents

List of Tables

List of Figures

Glossary

AAL ATM Adaptation Layer.

ACK Acknowledgment.

API Application Program Interface.

ASA Adaptation Service Agent. The process (thread) performing MAS functions.

ATM Asynchronous Transfer Mode.

ATMARP ATM Address Resolution Protocol. This protocol handles the conversion from IP address to ATM address and reverse.

CTS Common Transport Sublayer. A sublayer in QRTP layer which performs transport functions for QRTP TPDUs. It is under Media Adaptation Sublayer.

IETF Internet Engineering Task Force.

MCA Master Control Agent in QRTP implementation.

MAS Media Adaptation Sublayer. A sublayer in QRTP layer which performs media adaptation to and from QRTP TPDUs.

MIB Management Information Base.

OC-3 Optical Carrier 3. A SONET standard. Bandwidth is 155 Mbps.

PCB Protocol Control Block.

PVC Permanent Virtual Circuit. As opposed to SVC, PVC is established manually by using ATM network management tools.

RSVP Resource Reservation Protocol. A standard for IP layer resource reservation proposed by IETF.

RTP Real-time Transfer Protocol standard proposed by Audio/Video Work Group of IETF.

QRTP Queen's Real-time Transport Protocol.

SNMP Simple Network Management Protocol.

SONET Synchronous Optical Network. A North America standard on optical transfer of signals. It includes different rates such as 52 Mpbs (OC-1), 155 Mbps (OC-3), 465 Mpbs (OC-12). It is in Physical Layer in OSI Reference Model.

SSRC Synchronized Source. See definition in IETF RTP.

SVC Switched Virtual Circuit. SVC, as opposed to PVC (Permanent Virtual Circuit), is established automatically by programs sending ATM setup messages to network.

TPDU Transport Protocol Data Unit.

TSA Transport Service Agent. The process (threads) performing CTS functions.

VC Virtual circuit at ATM layer.

Chapter 1

Introduction

This thesis is motivated by the development of ATM (Asynchronous Transfer Mode) network technology and the demand for multimedia applications. As a switching technology, ATM presents enormous advantages over the traditional network technology such as Synchronous Transfer Mode (STM) and traditional LAN (Local Area Network) technology [Hui94]. Great development effort invested in both ATM technology, and underlying physical layer technologies such as SONET (Synchronous Optical NETwork), has secured the reliable and efficient high-speed services offered by ATM networks.

One of the advantages of ATM is that the single switch architecture can handle data cells of different media types. This feature presents a great opportunity for multimedia application development and deployment. On the other hand, demand for applications which utilize video, audio, high-definition images, and real-time data has increased with the growing connectivity of Internet and Intranets.

The applications which interest us the most are those involving video and audio, such as video conferencing and video-on-demand (VOD). Traditional network protocols, such as TCP/IP can be used to develop such applications, however inefficiencies exist. It is yet to be seen whether IP Next Generation (IPng) [DH96] over ATM, which is still under development, will be ideal for multimedia application development over ATM. Our research looks into constructing a real-time transport protocol which communicates over ATM network and conforms to the Real-time Transport Protocol (RTP) standard.

1.1 Problem Statement

As ATM becomes widely deployed as a broadband network technology, support for application development is becoming an important issue, especially for applications such as video and audio conferencing, which involve real-time data transfer. Development of these applications requires a middleware which functions as a transport protocol, and which provides an easy-to-use application programming interface (API). Such a protocol was first proposed by the Internet Engineering Task Force (IETF) [IETF-AVT95] as Real-time Transport Protocol (RTPV2). RTPV2 was implemented over TCP/UCP/IP as a user-extended transport protocol in various audio and video tools. However, RTPV2 has not been implemented over ATM. Therefore, one of our targets is to construct a basic software structure which implements RTPV2 over ATM.

Once RTPV2 has been implemented, the next challenge is to identify the main elements which control the delivery of real-time services to applications. Quality of Service (QoS) is a core concept in ATM technology and has been clearly defined in

ATM standard specifications [ATMFORUM-UNI96, ATMFORUM-PNNI96]. However, how to provide QoS from the higher layer is still a research issue. We believe that resource reservation and dynamic control of service entities, including scheduling, are in most part, sufficient for most real-time applications. The challenge is therefore to implement certain resource management and dynamic control algorithms and to evaluate their effectiveness.

Application development is also an important component of the research since it is vital to the testing of the function of the transport protocol middleware. We have selected video and audio conferencing tools as our primary test applications. The method employed to measure the performance of the middleware is also important, since it must allow us to not only collect performance-related information on a timely basis, but also analyze the impact of the information collection process on the performance.

Other research issues addressed in the thesis are the following:

- Synchronization between different information flows, which is a requirement presented by some real-time applications such as audio and video conferencing.
- Multicasting, which is required in a multi-user environment.
- Flow control, which is associated with the performance of the transport protocol.
- A management protocol, which is required to manage and control the operation of the transport protocol. In particular, we look at using a standard protocol such as Simple Network Management Protocol (SNMP).

1.2 Goals of the Research

Based on the problems and issues presented above, the goals of the research are the following:

1. Design and implement a basic set of software modules on top of the ATM network layer which provides real-time transport services to applications.
2. Evaluate the performance of this protocol with respect to meeting the requirements of typical real-time applications.
3. Study the other issues involved in real-time broadband application and extend the transport protocol to provide solutions to these issues.

The underlying assumptions for the research are the following:

- The ATM network can provide reliable broadband network communication (so retransmission can be omitted for real-time purposes).
- Development and testing are based on single processor workstations, Sun Sparc 4's, with real-time thread support in the operating systems, for example, Solaris 2.5.
- The network is a simple ATM network, with 2 or 3 Sun Sparc 4 stations connected to 2 Newbridge 36150 ATM switches via OC-3 interfaces [1]. 140 Mbps connections are used between switches.
- Video and audio programming are based on high-level interfaces offered by the Solaris 2.5 operating system, such as the XIL library, instead of any low level media manipulation interface.

[1] OC-3 is an optical fiber networking standard with bandwidth of 155 Mbps.

- Fixed size protocol data units are used.

In this thesis we will present an architecture, called QRTP (Queen's Real-time Transport Protocol), which has the following features:

- An easy-to-use API.
- A hybrid addressing scheme which uses IP addresses to initialize or discover session endpoints, and uses ATM addresses for sending payload data directly over the ATM stack.
- A standard protocol data unit format for the Internet community, which will provide future compatibility with other IETF AVT (Audio Video Transport) applications.
- A layered design which provides low protocol data unit loss and maximum guaranteed processing delay.
- A multi-entity concurrent processing model which further ensures better QoS under heavy transport load.
- A clearly defined management scheme and management information base (MIB).
- Flexibility in the use of middleware because it can be tailored for needs of different applications.

1.3 Outline of the Thesis

The thesis is organized as follows. In Chapter 2 we provide background to the thesis. It presents terminology and basic concepts and then discusses related work in the

areas of ATM, IP over ATM, and development support for broadband applications. We next summarize the functional requirements of a real-time transport protocol, and its relationships with upper level applications and low level networks, then the design details in Chapter 3. Chapter 4 describes the protocol mechanisms. Chapter 5 describes the implementation. Implementation of the middleware is done in C, using Solaris threads. Based on the middleware implementation, two applications which are used to test the package were also developed. Application development is briefly described in Chapter 6. In Chapter 7 a performance evaluation of the middleware is presented. The concluding remarks, a summary of the contribution of this work, and future research directions are given in Chapter 8.

Chapter 2

Background and Related Work

Broadband multimedia communication is developing at an unprecedent rate. Both industry and research institutes are working towards producing more services by taking advantages of the bandwidth made available by ATM networks. While the network architecture has been the focus of standardization, end-point architecture design has been left to different service providers. As a result, different protocol stacks have been developed for different target domains. Such protocols include HTPNET [CG94], IRM (Integrated Reference Model), OPENSIG (Open Signaling) [Laz92], and the Tenet protocol suite [BFMMVZ94]. However, in the long run, these structures will converge as different computer network services and telecommunication services converge, because they all use the same underlying network structure. Before we study the entities related to the real-time transport protocol, it is necessary to paint an overall picture of the current components of broadband multimedia services, their relationships, and where our research fits in.

2.1 Terminology and Basic Concepts

This section presents terminology and basic concepts for real-time multimedia applications over broadband networks.

2.1.1 Real-time Transport Service

We define a *real-time transport service* as a service provided by the transport layer (OSI Reference Model Level 4) to distributed multimedia applications. This service is characterized by (1) fast and time-bounded data delivery; (2) guarantee of event temporal characteristics during playbacks; (3) synchronization between different data flows; (4) support of multicasting for a multi-user environment; (5) good buffering and flow control mechanism to reduce data loss, and (6) QoS monitoring and control mechanisms.

- *Fast and Time-bounded Delivery.*

  ATM technology implemented on the top of SONET can provide high bandwidth (155 Mbps in case of OC-3) to allow traffic from multiple applications, each of which can reserve some bandwidth, to be multiplexed onto the same connection. ATM supports *statistical multiplexing*, which means that ATM access equipment can multiplex traffic onto virtual circuits and statistically, rather that constantly, guarantee the quality of services to each traffic source. Therefore, although an ATM network can provide fast delivery, in order to achieve time-bounded real-time services, a transport protocol must perform two tasks: (1) present proper QoS requirements to the ATM networks, and (2) implement a transport layer QoS control and resource reservation mechanism to ensure that the delay time within the transport layer is bounded. In our research, for

any payload data sent over an ATM network, we select the Constant Bit Rate (CBR) class for ATM QoS because it ensures a real circuit-like connection with dedicated bandwidth [ATMFORUM-TM95].

- *Temporal Characteristics in Playbacks*

  For real-time applications, it is important to preserve two characteristics of an information flow, namely, the order of the events, and the time interval between any neighboring events. For example, in video conferencing, if the remote participant makes one move followed by another move 10 seconds later, then in the playback the time interval between these two moves should also be 10 seconds. This requirement means that not only must the data representing the event be delivered to the remote end point within the time boundary, but it must be delivered to the application for playback at the *exact* time.

- *Synchronization*

  Synchronization means that the events in one data flow are to be correlated with those in another. An example of synchronization is lip-sync in video-conferencing, where voice stream flow is synchronized with video stream flow. In our research we consider synchronization between two data flows.

- *Buffering and Flow Control*

  The ATM network layer performs very little flow control for the class of QoS we have selected. Flow control is left to the transport or application layer. At the transport layer, flow control ensures that the receiving entity can accept all the protocol data units transmitted by the sender, and that loss is reduced to a minimum.

- *QoS monitoring and Control*

  As defined by Stallings [Stallings93], performance management comprises two functional categories–*monitoring* and *controlling*. In a real-time transport services environment, it is important for users to monitor the operational status. Also, it is ideal if an intelligent management application can automatically monitor the services and assert control decisions on the services. Performance related information, mostly equivalent to the QoS information which user applications are concerned about, should be organized into proper groups and presented by the management system in a clear format.

2.1.2 Broadband Applications Over ATM

There are a wide range of broadband applications currently being developed in both industry and academic research institutes which use broadband networks for new services. Examples are residential broadband services such as video-on-demand (VOD) and video shopping, and business broadband services such as desktop video conferencing, telerobotics applications and telemedicine. Broadband applications can be broken into two categories based on the communication directions: (1) *conversational services* and (2) *presentational services.*

- Conversational services such as video conferencing involve symmetric communication among the parties. The QoS requirements presented by all applications endpoints are the same.

- Presentational services such as VOD involve asymmetric communications, with a large amount of bandwidth from the presenter of the information to the receiver (downstream) and a small bandwidth being used for upstream.

Applications can also be categorized based on the number of participants in the application operation. We can identify two categories: (1) *unicast services*, and (2) *multicast services*.

- Unicast services involve at most two parties in a point-to-point communication.
- Multicast services involve multiple endpoints in a point-to-multipoint or multipoint-to-multipoint communication. This requires the transport services to offer capabilities such as: (1) the ability to allow application endpoints to join, withdraw from an established information flow, and (2) the ability to deliver copies of the same data to all participants of a session.

In our work we have attempted to construct the transport service protocol such that it can be used to serve all categories of applications but we have only built conversational multicasting test applications.

2.1.3 Quality of Service

There are many different metrics which can be used to indicate the performance of different applications. For example, Nahrstedt gives a set of audio and video QoS parameters and values [NAHR95]. We define transport layer QoS to include metrics which are common for most applications, namely, end-to-end delay, inter-arrival jitter and error rate.

1. *Bandwidth (BW)* is the number of TPDUs (Transport Protocol Data Units) that transport entities can process per second.
2. *End-to-end delay (EED).*

   EED is the time between a TPDU entering the QRTP transmitting entity, and

the same TPDU departing the QRTP receiving entity at the remote end. This is made up of the following components: (1) queuing delay at the QRTP sending entity, (2) processing delay at the QRTP sending entity, (3) transmission delay from local ATM endpoint to remote ATM endpoint, (4) queuing delay at the remote QRTP receiving entity, and (5) processing delay at the remote QRTP receiving entity. Since we use a fixed size protocol data unit, (2), (3) and (5) are constant, and the important variables are (1) and (4), which change depending on traffic load.

3. *Inter-arrival jitter* is the difference between the smallest inter-arrival interval and the largest one.

   This is important because we are considering constant rate media for real-time applications. If the inter-arrival jitter at the sender is 0, but it is large at the receiving end then the perceived QoS to the user is degraded. Therefore the QRTP receiving entity should perform some buffering and timing when forwarding received TPDUs to applications.

4. *Error rate* is the percentage of TPDUs lost or received with error.

   For some real-time applications such as audio conferencing and important data fetching, a high error rate will result in unacceptable results.

For each metric, an application endpoint presents a required value and a bound relative to the required value which can be translated into ATM layer QoS, traffic descriptor and bearer descriptor, and presented as part of ATM signaling requests when setting up the connection for information flow. As well, the metrics will be used by QRTP entities for QoS control at the transport level.

2.1.4 Resource Reservation and Scheduling

As we stated earlier, proper resource reservation and scheduling are critical for QoS control so the concepts of *resource*, *resource reservation* and *scheduling* have to be defined and the relations among these concepts and QoS have to be studied.

We define *resources* in this research to include (1) endpoint system CPU processing power (measured in number of TPDUs per second); (2) access to network (ATM) interfaces for sending and receiving data; and (3) buffer space for holding TPDUs waiting to be processed. *Resource reservation* is composed of two actions: (1) using the ATM signaling interface to reserve ATM layer resources for the transport layer connections, and (2) associating resource reservation information with the data structure that represents the transport layer connection. This structure is used in QoS control when the connection is operational. Resource reservation can be denied if the endpoint system internal information shows there are not enough resources available to meet the requirements.

Scheduling is using the reservation information on endpoint system CPU processing power to arrange the serving of different connections in order to meet the QoS requirements set when these connections are initialized.

2.1.5 Session

Session is the terminology used by [IETF-AVT95] to represent a transport layer RTP connection between applications. We extend this definition so that a session is a

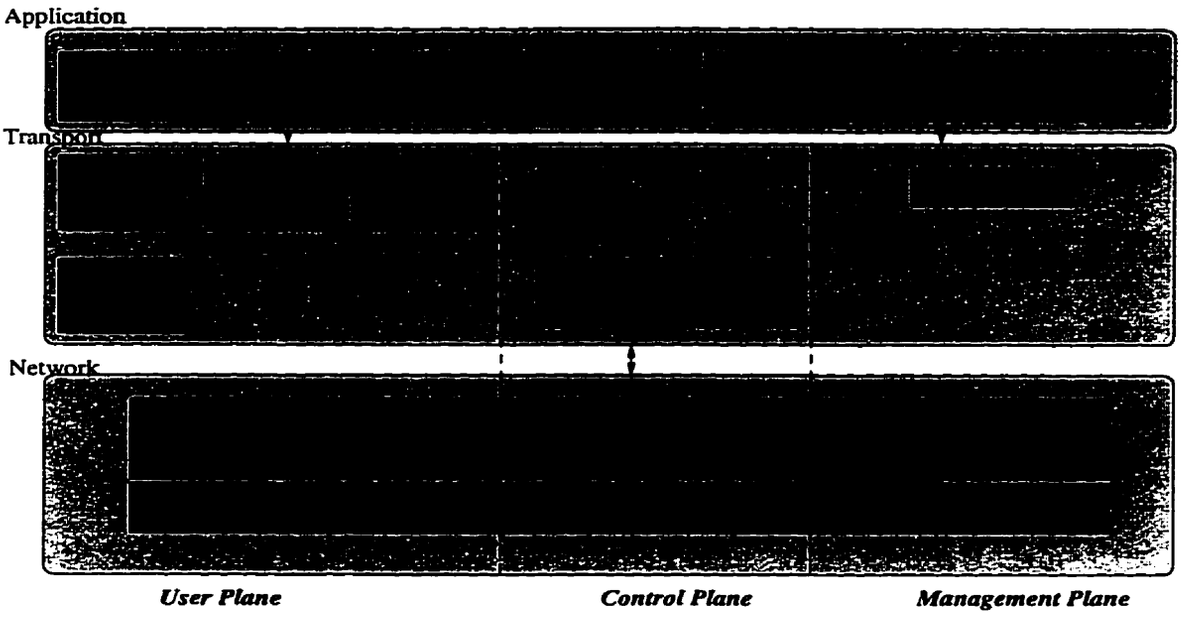

Figure 2.1: Overall state of broadband application protocol stack

transport level (OSI Level 3), end-to-end or multiparty connection, with certain resources dedicated to it. Two types of data are sent over sessions: *control data*, which includes session establishment and destroy, and user join and withdraw messages and *payload data* which is the data actually provided by the application services.

2.2 State of Broadband Multimedia Communication: An Overview

Figure 2.1 illustrates the overall protocol stack structure of broadband applications. Broadband systems are usually separated into three planes: *Control*, *User* and *Management* [Stallings93]. Broadband application research began in the telecommunication sector with the goal of providing new services to their customers. Applications such as Audio/Video Multimedia Service (AMS), Circuit Emulation Service (CES),

and Voice and Telephony over ATM (VTOA) are still in the works [Dob96]. As ATM starts to serve as a switching technology in a scale ranging from backbone telecommunication network to desktop workstations, distributed computer applications, multimedia and telecommunication applications start to converge. In addition, digitized multimedia development, transfer, and storage present more advantages and new services, so more multimedia applications are based on data communication stacks over ATM.

Most of the distributed applications run on the TCP/IP networks. IP over ATM was quickly designed and adopted after ATM technology was deployed. Some issues remain in the TCP/IP/ATM stack. One of them is the transmit-acknowledge cycle implemented by TCP [Clark92]. Since ATM provides high bandwidth, an acknowledgment-based cycle delay will waste network bandwidth. Chan et. al. also pointed out this problem [CS96] and adopted a direction of implementing Long-Fat Network (LFN) extension in workstations to increase buffer space to solve it. Other researchers take the view that a light-weight transport protocol which omits functions such as acknowledgment and error checking is needed for broadband applications. RTP (Real-time Transport Protocol) [IETF-AVT95] and XTP (Xpress Transport Protocol) are examples of this approach.

IP over ATM was standardized [IETF-IPATM95] by the Internet Engineering Task Force (IETF) and it was designed to internetwork an ATM network with an IP network, and provide TCP/IP stack support for ATM networks. An advantage of using TCP/IP for broadband application development is that applications are portable

since the TCP/IP socket interfaces are standard. A major disadvantage of using TCP/IP is inefficiency due to the fact that the ATMARP (ATM Address Resolution Protocol) table or cache has to be consulted for each IP packet to translate IP addresses to ATM addresses. Although IP over ATM has its disadvantages, it has been implemented on most of the existing ATM access products, such as access switches and adaptor cards, and most still believe IP can be a core network service in broadband networks. IP Next Generation (IPNg) [DH96] work is expanding IP into a multimedia network service with extended address space. IP multicasting solutions are considered to compensate for the lack of multipoint-to-multipoint connection in ATM network [All95].

The signaling protocol is an important component of a broadband communication network. *Signaling* is the process of establishing connections (or virtual circuits) and reserving resources. For ATM, both user-network and network-network signaling have been standardized [ATMFORUM-UNI96] [ATMFORUM-PNNI96]. However, how IP endpoints signal connections between them is still an open issue because the current IP connection set up interface does not support resource reservation features. RSVP[BZB93] is a first step toward solving it. RSVP was designed to give IP signaling and QoS capability. ST2 was designed as an alternative to RSVP and it is easier to map from ST2 to ATM than mapping from RSVP to the ATM [Jac96]. ATM API (Application Program Interface) is still an issue that needs to be standardized [Dob96] and questions remain whether native ATM support or IP over ATM should be accessed by ATM API.

2.3 Work in Broadband Applications

To provide a typical example of interactive conversational multimedia applications, IETF has implemented a suite of audio/video multimedia conferencing tools over RTP (Real-time Transfer Protocol) [Sch96]. In this suite there is a master control tool `sdp`, which discovers on-going audio and video sessions. From within `sdp`, `vic` can be activated for video conferencing sessions, and `vat` can be activated for audio conferencing sessions. RTP headers carry vital real-time conferencing information such as timestamp, sequence number, and a list of sources which contribute to payload data. One shortcoming of these RTP implementations is that RTP is not implemented as a separate layer or module. Instead, RTP code is embedded in application code so reusability is compromised.

There are many multimedia presentational applications currently being developed, including video on demand (VOD), video shopping, and an experimental multimedia news service model based on broadband networks [Ooi95]. Most of these applications can be abstracted into a multimedia database system, with users running database clients and content providers running database servers. It is different from traditional database transactions, however, in the sense that downstream data has to be continuous for smooth playback at the client end. This imposes performance requirements on both the database server and the transport protocol entities used to deliver downstream data. Media adaptation, which involves converting application-specific media that conform to transport requirements to network packets that conform to network requirements, is an important component in the development of applications.

2.4 Work in Related Protocol Stacks

The goal of the research in protocol stacks, or middlewares, is to support broadband multimedia application development. Some of the research projects in this area include the following:

- *kStack.* kStack [AKS96] is a research very similar to ours. It builds a transport protocol over native ATM stack, to support general applications (rather than real-time applications). It is implemented on personal computers (PCs) with emphasis on QoS control and task scheduling. Interrupt and interrupt service routines (ISRs) are used for synchronous service of TPDUs. Performance measurement from kStack can be used to be compared against our evaluation.

- *RTP*[IETF-AVT95] is on-going work within IETF's Audio/Video work group to define a real-time transport protocol format for packetized audio and video delivery over the Internet. Conferencing applications using it have been developed and are in wide use.

- *Tenet protocol suites*[BFMMVZ94, BFG+95] is a suite of transport protocols including *Real-time Message Transport Protocol* (RMTP) and *Continuous Media Transport Protocol* (CMTP) which run over the *Real-time Internet Protocol* (RTIP). Rate control and deadline-based scheduling, which are mathematically proven to provide deterministic and statistical QoS bounds [BFG+95, BM91], are implemented. This suite is running on a few ATM testbeds. But to the best of our knowledge there is no report on applications developed using Tenet.

- *OPENSIG*[Laz96, Laz92] produced the *Integrated Reference Model* (IRM). IRM functions are carefully separated into 5 planes and each plane is layered according to its functions. The *User Information Transport* plane (U-plane) handles payload transport and uses *Intelligent Multiplexers* (IM) to control the service of link buffers by links. IM in turn uses a resource and QoS control database maintained by other planes, such as the *Resource Monitoring and Management* (D-plane) and the *Resource Management and Control* (M-plane), to make intelligent decisions on scheduling of services.

- *The Xpress Transport Protocol (XTP)*[XTP95] consortium conducts research into implementing Xpress Transport Protocol (XTP), which has a variety of features such as QoS negotiation, selective acknowledgment, explicit multicasting, message scheduling and concurrent service model. An XTP header includes control commands (common, error and traffic) and control information (address, traffic, diagnostic) that can be configured and tailored to user applications. The disadvantage is that XTP is overly complicated for real-time, fast data transfer. XTP has been implemented by researchers at the University of British Columbia [MN95].

- *IP over ATM* [IETF-IPATM95] is an IETF working effort [IETF-IPATM95] to establish Internet Protocol (Version 4) over ATM network, so that ATM can be used as a data link layer. Main issues that have been solved include the address resolution protocol (ATMARP), signaling of virtual circuits by IP entities [PLMHGM96], and support of multicasting [Arm95]. However, it still remains how IP over ATM can utilize the QoS features presented by ATM. That is, there has to be IP level signaling mechanism which supports resource

reservation and real-time QoS guarantee [BCB+95]. Both RSVP and ST are being considered for that purpose.

- *RSVP* (Resource Reservation Protocol) [BZB93] was designed and implemented in experimental scale to support IP level QoS signaling. A User can employ the RSVP API [BH95], or command interface, to specify QoS requirements for a specific IP flow. The resource reservation is *receiver-based*, which means each receiver specifies the QoS it obtains when the data is delivered. QoS is either *controlled* or *guaranteed*, depending on the level of priority. The RSVP daemon works with the IP packet filter to separate packets into different classes, and works with the IP packet scheduler to provide different QoS to different classes.

- *ST2* and *ST2+* are both IETF efforts to define an IP protocol which supports resource reservation signaling and QoS guarantee in transmission. Native ATM support for ST2 and ST2+ is more logical and easier than the combination of IP over ATM and RSVP [Jac96]. ST2+ contains two protocols: SCMP for control and signaling, and ST for payload delivery.

2.5 ATM API and Protocol Development Environments

An Application Program Interface is important for higher layer entities in order to provide access to services offered by a lower layer. The ATM Forum is currently working towards standardizing an ATM API [Dob96]. The on-going effort includes

- *IBM ATM API.* IBM has produced a socket extension for native ATM access [HSC+96]. Programmers use the standard socket interface but with new parameters for accessing ATM services. Rate-controlled connections are supported. Applications can specify ATM QoS parameters and issue switches virtual circuit (SVC) signaling requests.

- *Fore System API* [Fore96]. Fore Systems offers two types of ATM APIs. One is a TCP/IP/ATM socket API for legacy TCP/IP applications, and the second is for accessing ATM native services. Socket interfaces are exactly the same as the traditional one. Native ATM API allows applications to use SPAN (Signaling Protocol of ATM Network), which is compatible with UNI 3.1 [ATMFORUM-UNI96], to issue signaling requests to the ATM network. In addition, there are API calls allowing applications to send and receive data over an ATM network.

- *x-Kernel* [MO95] is a protocol construction tool based on a concept of dynamic protocol stack building process. It provides a framework for protocol designers to define the protocol stacks and then compile the stack into a set of C files (including the main program of the protocol service daemon), to which the protocol implementation code can be added. Code generated by x-Kernel handles all the interactions between protocols (queues, buffers, etc.). It also provides the implementation of most popular protocols such as IP, TCP, and UDP. Developers can expand the pool of protocols and combine them within certain limits. x-ATM [TC96], developed at University of Illinois, is an extension of x-Kernel which includes some vendor specific ATM interfaces. Using both toolkits, new transport protocols can be designed to utilize existing low

level protocols and fit specific needs. Another contribution of x-Kernel is the concept of using a concurrent service model of "one-thread-per-message" which effectively increases the performance and reduces packet loss. A shortcoming of both x-kernel and x-ATM is that, when developing an application, application code is built into the same program with the lower protocols. This means every application instance contains a complete protocol so centralized control of protocol service is impossible. A way around this problem is to use the protocols to implement a daemon, and then design a message-based interface which all applications use to communicate with the daemon. This, however, violates the x-Kernel concept of only using the provided inter-layer interfaces.

Summarizing the research effort on ATM API, there are three directions: (1) extension of existing socket APIs, (2) development of new ATM APIs, and (3) development of integrated toolkit which integrates other protocols into a single stack. It is yet to be seen which one ATM Forum favors.

2.6 QoS

Nahrstedt uses Anderson's theoretical result [And93] to study QoS guarantee with proper resource management and scheduling [NAHR95]. Internet Engineering Task Force (IETF) Resource Reservation Work Group has proposed Resource Reservation Protocol (RSVP) [BZB93] as an IP signaling protocol which reserves resources to provide controlled or guaranteed quality of service to distributed applications. Other researchers have also performed similar research.

2.7 Summary

A large effort has been put into research of both broadband applications and real-time protocols. However, no standards or guidelines have been produced. IETF's RTP is a good start for standardization, and it is necessary to produce a generic RTP implementation over ATM. QoS and signaling are concerns in designing any such protocol. Resource reservation and service scheduling at end systems are the keys. End system performance, buffering and flow control are important when the network bandwidth is to exceed end point system processing bandwidth.

Chapter 3

QRTP Architecture

In this chapter we present the design of the Queen's Real-time Transport Protocol (QRTP). The goal of the design is to produce a protocol architecture which supports real-time communication over ATM networks. It must (1) support a connection-oriented communication model, in which end-to-end connection is set up before payload communication commences; (2) support multicasting and group management; (3) allow applications (users) to specify QoS requirements for the connections; (4) have good overall performance and low loss rate; (5) support constant bit rate (CBR) communication; (6) provide an easy-to-use API for accessing the services, and finally, (7) utilize native ATM services efficiently.

The design goal in turn leads to a number of important design decisions. We seek to improve the performance of the protocol entities by using a concurrent service model, based on real-time UNIX threads. We also choose to use a vendor specific ATM API to access native ATM services, bypassing the IP/ATM stack. The standard IETF RTP V2 protocol data format was selected as the packet format because of its

real-time information, and the flexibility of allowing user-extended headers.

3.1 Protocol Data Format

The RTP V2 protocol data unit (TPDU) format [IETF-AVT95] has been specified by IETF's Audio/Video Transport Work Group. It contains information such as payload type (for example, MPEG1), sequence number (for reconstruction purposes), transmission timestamp (for real-time control purposes), list of parties contributing to the payload (in case of multiplexing media from different sources into the same TPDU), and source identification. We extend the header, by using the *header extension* portion, to include implementation-specific information such as receiving timestamp (for receiver performance evaluation), window size (for dynamic definition of window size in selective acknowledgment), packing flag (to indicate whether multiple payload units are to be packed into the same TPDU to save processing time), and number of payload units packed if packing flag is set.

Currently we use a fixed TPDU size of 3000 bytes for easy and fast receiver processing. We decided *not* to segment (reassembly) TPDUs into (from) smaller units because the AAL layer already partitions data into 48-byte ATM data cells and performing segmentation and reassembly in the transport layer could seriously degrade the end-system performance.

We illustrate the TPDU format in Figure 3.1. In our design, we have a total header length of 100 bytes (including extended header), which is an overhead of 3%. The description of the original RTP header fields are found in the RTP specification

[IETF-AVT95]. The contents of the RTP header extension are explained below:

- *Window size.* Due to the real-time nature of applications, and the large delay bandwidth product in broadband networks, not all data packets can be acknowledged because it wastes too much bandwidth. However, selective acknowledgment is optional for flow control purposes. We adopt a policy which sends back one ACK for each window of packets, where the size is defined by this field.

- *Transmitting timestamp (ms).* Although the original specification includes a timestamp in the header, it is hard to keep both the seconds portion and the microseconds portion of the timestamp in the same field. In our implementation timestamps are at a microsecond precision to facilitate precise measurement of queuing delay at sender end.

- *Receiving timestamp*, a timestamp similar to the transmitting timestamp above, but at the receiving end.

- *Packing flag.* 1 if packing occurs, 0 otherwise.

- *Packed number of units.* If packing flag is 1, then this field contains number of payload units packed into a TPDU.

- *Payload size.* Allow the user to redefine the packet size.

- *M/U Flag.* This binary flag, which is a bit, identifies whether a TPDU is for multicasting or for unicasting. When a user sends TPDU to session owner, it can be either unicasting from the user to owner, or multicasting to all users. Owner service agent makes decision based on this flag.

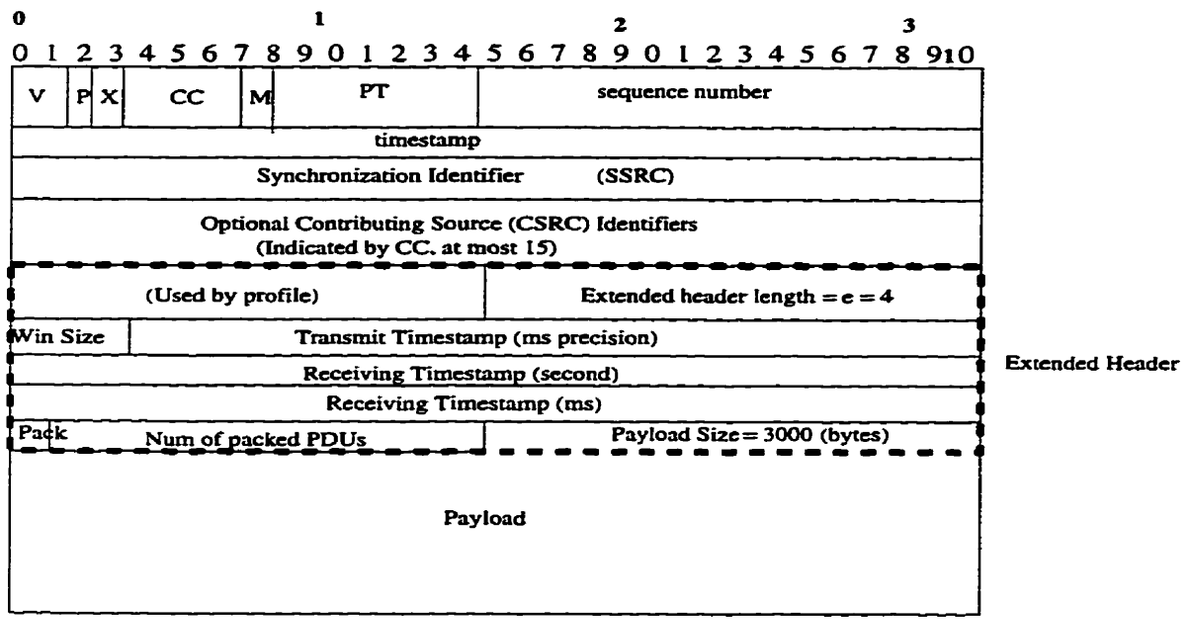

Figure 3.1: Protocol Data Unit

3.2 Protocol Middleware Structure

The overall structure of the QRTP middleware is illustrated in Figure 3.2. It is a separate layer, which lies between the application and network layers, and is vertically separated into three planes: the *Control Plane*, *Management Plane* and *User Plane*. The Control plane has one entity, the *Master Control Agent* (MCA), which communicates control traffic from (to) applications to (from) the lower layers and ultimately to (from) remote ends. Horizontally, QRTP is separated into two layers: the *Media Adaptation Sublayer* (MAS) and the *Common Transport Sublayer* (CTS). At a sender endpoint, MAS sublayer entities, which are called Adaptation Service Agents (ASAs), convert user media payload into one or more RTP TPDUs, and enqueue them onto the TPDU queues. CTS entities, which are called Transport Service

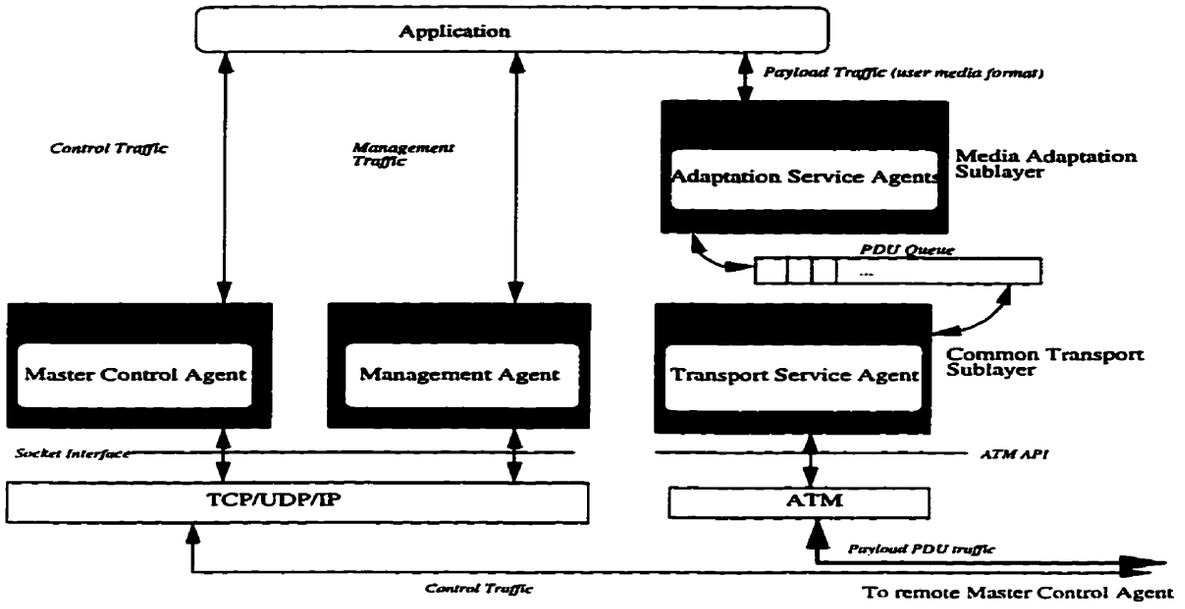

Figure 3.2: Overall structure of QRTP

Agents (TSAs), dequeue the TPDUs and send them to the remote end via the corresponding ATM virtual circuit, which is established when the session is established. At a receiver endpoint, the same process takes place in the reverse order. In one endpoint system, there is only one MCA and MA, but there are multiple ASA/TSA pairs. Each ASA/TSA pair serves one or more sessions.

There is an important global data structure, called the *Protocol Control Block* (PCB), for every session. It is used in the TCP/IP implementation [WS95] and we adopt a similar structure for storing connection-oriented session control and management-related information. This information includes session identification, list of users, underlying ATM information such as VCI/VPI (Virtual Circuit ID and Virtual Path ID) pair, and most importantly, performance information. It is used by an MCA to assert control on different sessions (such as scheduling). It is also used by service

agents (such as ASAs and TSAs) in serving sessions, and by management agents to provide vital management information to remote managers.

Such a design has the following important features:

- Interactive communication between session owner and session users.
- Transport layer point-to-multipoint and multipoint-to-multipoint communication.
- A concurrent service model which allows easy scheduling of services for individual sessions for real-time services.
- A clear separation of control, management and user functions.
- Utilization of the native ATM stack for payload delivery.
- A clear separation of the user media adaptation function and the TPDU delivery function, and a variable size queue to pass payload data between them.
- A receiver endpoint ASA handles most of the real-time functions such as resequencing while the TSA handles simple receiving of TPDUs in order to reduce TPDU loss rate.

3.3 Protocol Entities

3.3.1 Master Control Agent

The Master Control Agent (MCA) is the main driver in an endpoint. It handles tasks such as (1) establishing or destroying a session when requested by a local application (*Session_Init* or *Session_Terminate* request); (2) adding or removing a remote or local application as a user of an existing session when requested by a remote MCA or local application (handling *Session_Join* or *Session_Withdraw* request); (3) sending requests on behalf of local application to a remote MCA to join or to be be removed from, a remotely-initialized session (handling *Session_Join_Remote* or *Session_Withdraw_Remote* request and sending *Session_Join* or *Session_Withdraw* requests to remote MCA); (4) controlling and managing of all other service agents on the endpoint system; (5) obtaining and assigning ATM Virtual Circuit IDs to sessions.

Event handling by the MCA is as follows:

- When the QRTP service starts, and an MCA is being initialized, the sequence of events is as follows. First, the communication channel between MCA and applications is initialized. Data structures such as the lists of sessions and users are initialized. All of these data structures are made global so that they are accessible by other agents. A Management agent (MA) is started and it begins listening for management query requests. Finally, the MCA listens for requests from both local and remote entities.

- When receiving a Session_Init request, the sequence of events is as follows: (1) The MCA compares the resource requirement information in the request with

the internal resource data structure. If there is not sufficient resources available, then the MCA responds with a negative response and aborts the initialization process, otherwise the MCA continues. This is called *admission control.* (2) The MCA creates a new session and the data structures representing it. (3) The MCA initializes communication channels for sending and receiving payload data. If ATM SVC signaling is used, then signaling requests are sent out. If an ATM PVC (Permanent Virtual Circuit) has already been provisioned, then connect the PVC with the MCA. In either situation, the MCA stores the VPI/VCI pair in the session data structure so that any subsequent payload data will be sent out over the specified VC. (4) The MCA creates two ASA and TSA pairs. One pair (T-ASA and T-TSA) for transmitting, the other (R-ASA and R-TSA) for receiving. The agents will use the global session information maintained by the MCA. (5) The MCA sends response information, including the information needed to communicate with T-ASA and R-ASA agents, to the application that is requesting the initialization. For any subsequent payload transport, the application communicates with the ASAs only. (6) The MCA hands the duty of serving of the session to agents then listens for new requests again. This process, along with the QRTP initialization, is illustrated in Figure 3.3.

- When receiving a Session_Join request, the sequence of events is: (1) If the request information indicates a local user, the MCA establishes a local communication path between ASAs of the session and the user, or (2) if the request information indicates a remote user, the MCA sends local session information, including the ATM VCI/VPI pair and ATM address to the remote end MCA.

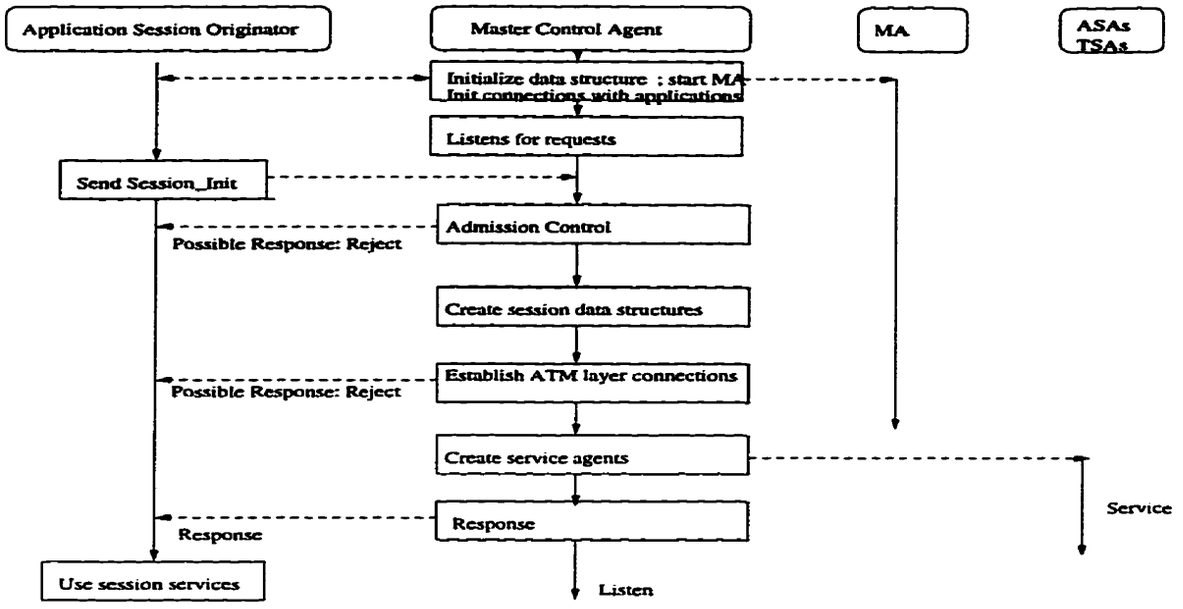

Figure 3.3: MCA behavior when receiving Session_Init request

The remote end MCA should issue signaling or a PVC connection request to connect local TSAs with remote TSAs. (3) The MCA adds the user information to the list of users for the session.

- When receiving a Session_Join_Remote request, the sequence of events is: (1) The local MCA translates the request into a *Session_Join* request to send to a remote MCA. (2) The local MCA sends a *Session_Join* request to the remote MCA then waits for a response. (3) A local MCA receives a response from a remote MCA, which contains connection information (such as ATM address, remote VPI/VCI pair) for remote TSAs. (4) A local MCA creates a new local session and the data structures representing it. (5) Communication channels for sending and receiving payload data is initialized. If ATM SVC signaling is used, then signaling requests that connect the remote TSAs (See (4)) with local TSAs are sent out. If ATM PVC is used and it has already been provisioned,

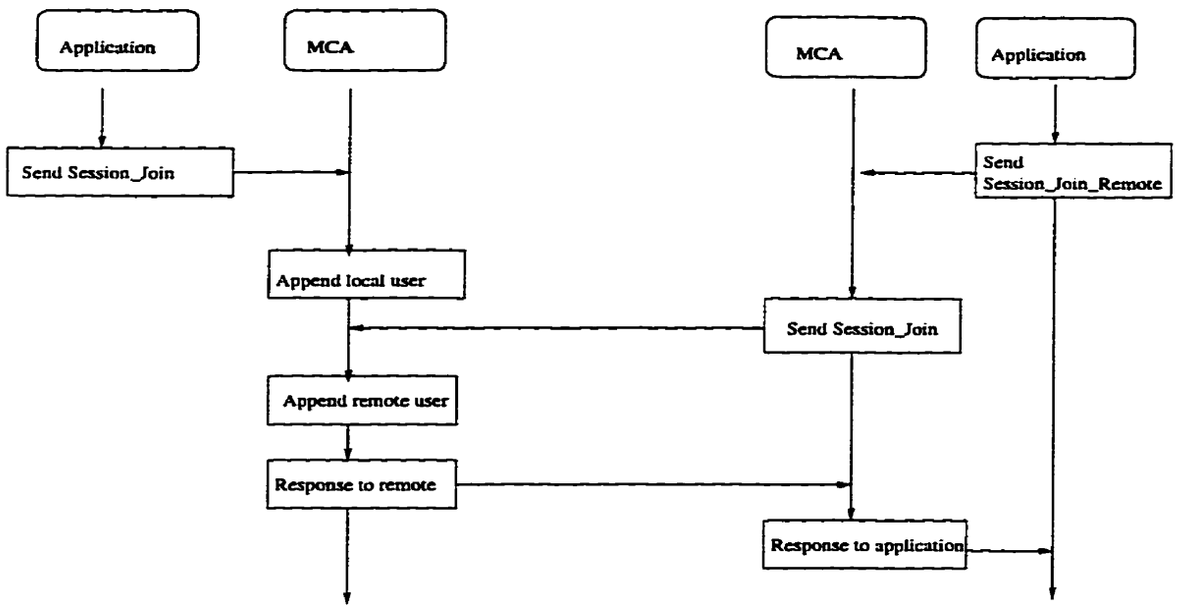

Figure 3.4: MCA behavior when receiving Session_Join and Session_Join_Remote request

then connect the PVC with MCA. In either situation, associate the VPI/VCI pair with the session data structure so that any subsequent payload data will be sent out over the specified PVC. (6) The local MCA creates two ASA and TSA pairs. One pair for transmitting (T-ASA and T-TSA), and one pair for receiving (R-ASA and R-TSA). The agents will use the global session information. (7) Local MCA sends response information, including the information needed to communicate with T-ASA and R-ASA agents, to joining application. The requesting application will contact only ASAs subsequently. (8) MCA hands the service of the session to agents then listens for new requests again. The above two cases are illustrated in Figure 3.4.

- When receiving Session_Withdraw request, a *Session_Withdraw* request is sent to appropriate remote MCA.

- When receiving Session_Withdraw_Remote request, identified user is removed from the session's user list.

3.3.2 Adaptation Service Agent

The behavior of a receiving ASA (R-ASA) is defined as follows: (1) If the receiving TPDU queue is not empty, dequeue a TPDU from the queue, (2) extract the user data from the TPDU, (3) perform real-time control functions such as out-of-sequence detection and recovery, and synchronization. (4) Go back to (1).

The behavior of a transmitting ASA (T-ASA) is defined as follows: (1) Accept user data, (2) construct TPDU which includes header (constructing TPDU following specification in RTP Profile [Sch96], referred as *adaptation* process), (3) enqueue TPDU onto the sending queue, (4) repeat (1) to (3).

3.3.3 Transport Service Agent

The behavior of a receiving TSA (R-TSA) is defined as following: (1) Receive TPDU from ATM layer, (2) enqueue TPDU onto the receiver TPDU queue. Functions of the R-TSA are deliberately kept simple since the ATM VC buffer is limited and in a heavily-loaded situation, the longer an R-TSA takes to perform its task, the more TPDUs that will be lost.

The behavior of a transmitting TSA (T-TSA) is defined as following: (1) If sender TPDU queue is not empty, dequeue TPDU, (2) send TPDU over the ATM VC which is used by the session.

3.3.4 Management Agent

Upon receiving management information query, Management Agent finds the session being queried, consults PCB for the session, extracts the information, constructs a response and sends the response to the querying management application.

Chapter 4

Protocol Mechanisms

Stallings [Stallings93] provides a thorough description of generic transport protocol mechanisms. His analysis of mechanisms are based on classes of network services. We follow a similar pattern.

The ISO defines three types of network service:

- Type A: network connections with an acceptable residual error rate and acceptable rate of signal failures. Within Type A, there are three subclasses:
  - Reliable, sequencing network service with arbitrary message size.
  - Reliable, nonsequencing network service with arbitrary message size.
  - Reliable, nonsequencing network service with maximum message size.
- Type B: network connections with acceptable residual error rate but unacceptable rate of signaled failures.
- Type C: network connections with residual error rate not acceptable to the

transport service user.

The ATM network based on a SONET physical layer, provides a reliable network service. Within the subclasses of Type A, ATM belongs to the one which features reliable nonsequencing network service with arbitrary message size. It is nonsequencing because in a wide-area network which might involve public carriers, there is no guarantee that end-to-end ATM traffic is not out-of-sequence. Although there is a guarantee that the AAL layer will reconstruct data packets from possible out-of-sequence ATM cells, there is no guarantee that the inter-packet order will not be out-of-sequence. Based on this observation, aspects of the protocol mechanism which must be discussed include: *addressing*, *multiplexing*, *Buffering*, *connection establishment and termination*, *resequencing*, *synchronization* and *multicasting*.

4.1 Addressing

The issue concerned with addressing in QRTP is simply this: a user of a given transport entity wishes to establish a connection (session) with a user of some other transport entity. We use a *hybrid addressing scheme*. That is, two steps are followed to establish the connection:

- End-points and services are identified by names or IP addresses, connections are established using IP address information, and
- payload transfer uses ATM addresses or ATM VPI/VCI pairs which identify the network connection.

Step 1 generates a mapping between session identifications and ATM VPI/VCI pairs, and a connection at the ATM layer within the network connecting end to end. The

ATM connection establishment can be automatic (using IP/ATM signaling), or manual (using network management tool). Step 2 uses the mapping to extract ATM connection information then uses the established connection to send or receive data. All these addressing details are hidden from applications.

The MCA on any host listens on a known port. When an application requests to establish a session or to join an existing session, the request message contains the IP address of the remote end(s) that it might want to communicate with, and session identification (if the request is about an existing session). The MCA uses the IP address as a parameter to call ATM signaling or connection binding functions to set up ATM connections between ends. The ATM address information, in the form of an ATM VPI/VCI pair (they are functionally equivalent), is stored in the global session information maintained by the MCA, and is readable by QRTP service agents (T-TSA and R-TSA). As a result, on each endpoint system, there is a one-to-one mapping between pair [IP address;session number] in the control plane and ATM VPI/VCI pair in the user plane. This process is illustrated in Figure 4.1.

The remaining issue is: before an application sends data to remote ends, how does it know the IP address of the destinations? The fact is that it does not have to. When an application requests a new session, a session identification is given. Subsequently as more remote endpoints join the session, an MCA maintains a list of users (participants) for each session. When the originator of the session sends data, it only has to indicate its session identification. It is the T-TSA's task to extract participants' address information from the session database maintained by the MCA

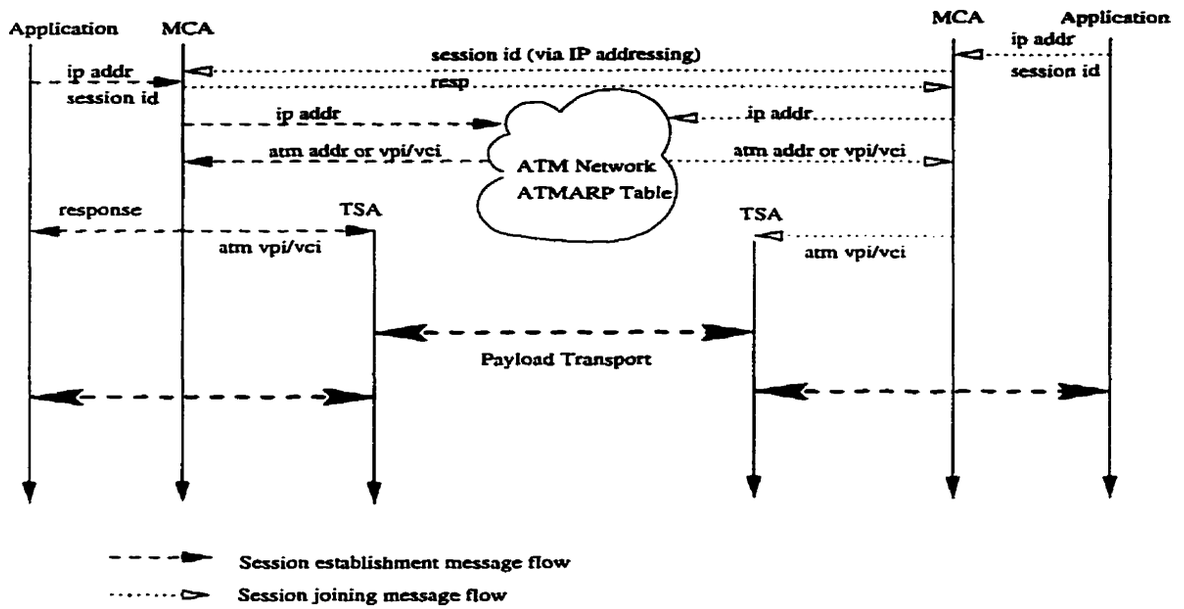

Figure 4.1: Hybrid addressing mechanisms

and deliver data to all participants on the list.

4.2 Multiplexing

There are three types of multiplexing: (1) *one-to-one* (no multiplexing), (2) *upward multiplexing* and (3) *downward multiplexing.* According to definitions given by Stallings [Stallings93], upward multiplexing occurs when multiple higher-level connections are multiplexed on, or share, a single lower-level connection. Downward multiplexing (splitting) means that a single higher-level connection is built on top of multiple lower-level connections.

Between an application entity and QRTP entities, it is either one-to-one or upward multiplexing. One-to-one occurs when one session is used to serve one application

entity. Upward multiplexing occurs when more than one applications are sharing a session. The later case imposes more difficulty on protocol design.

Between QRTP and lower layers, it is strictly one-to-one multiplexing, because the current ATM implementation does not allow multiple receivers to share the same VPI/VCI pair (therefore the same VC) at the same end point, although multiple senders can share the same VC.

As a result, we have a one-to-one mapping between [IP address;session number] pairs and virtual circuits at the ATM layer. A very important reason for such a decision, which is also described by [AKS96], is that multiplexing will lose the individual QoS specification of sessions involved.

4.3 Buffering

The purpose of flow control is to control the rate of transmission so that the receiving side buffer does not overflow and result in TPDU loss. A fundamental question is, *is flow control a desirable mechanism for real-time multimedia applications?* We conclude that it is not for the following reasons: (1) If flow control is employed, so that the sender entity is transmitting at a rate acceptable by receiver, real-time data is maybe lost at the sender side; (2) in a broadband network, if we employ any sort of transport layer flow control algorithm which involves acknowledgment, bandwidth will be wasted due to end-to-end delay and the large delay-bandwidth product. For example, an ACK message which takes 200 ms end-to-end delay in a wide area network will result in waste of 3000 bits (about 59 ATM cells) on a 15 Mbps virtual

circuit; and (3) flow control complicates the processing of both T-TSA and R-TSA and degrades their performance.

Our solution for TPDU loss is a combination of *fast R-TSA processing turn-around*, and *a variable size buffer*. R-TSA simply receives TPDUs and queues them, for further processing by an R-ASA. Also, our concurrent service model has the potential of employing multiple R-TSAs for a session under heavy load which further reduces the loss rate to a negligible level. A selective acknowledgment-based flow control algorithm is employed and it is optional.

There is a tradeoff between loss rate and play-back delay and it can be controlled by queue length. If we fix the queue length to a small value, then under heavy load a large number of TPDUs will be discarded at the R-TSA since there is simply no place to queue them. But for the TPDUs which do get queued and processed by an R-ASA, there is a short queuing delay, therefore there is a short playback delay between sender and receiver. This might be desirable for applications such as low-quality video conferencing, where quality of images is not as important as playback delay.

One the other hand, if we fix the queue length to be a large value, or we keep it variable, then all the TPDUs received by an R-TSA will be queued, even under heavy load. That may potentially result in longer queuing delay and therefore longer playback delay between sender and receiver. This is more desirable for applications whose speed is not as important as data integrity, such as medical data collection and

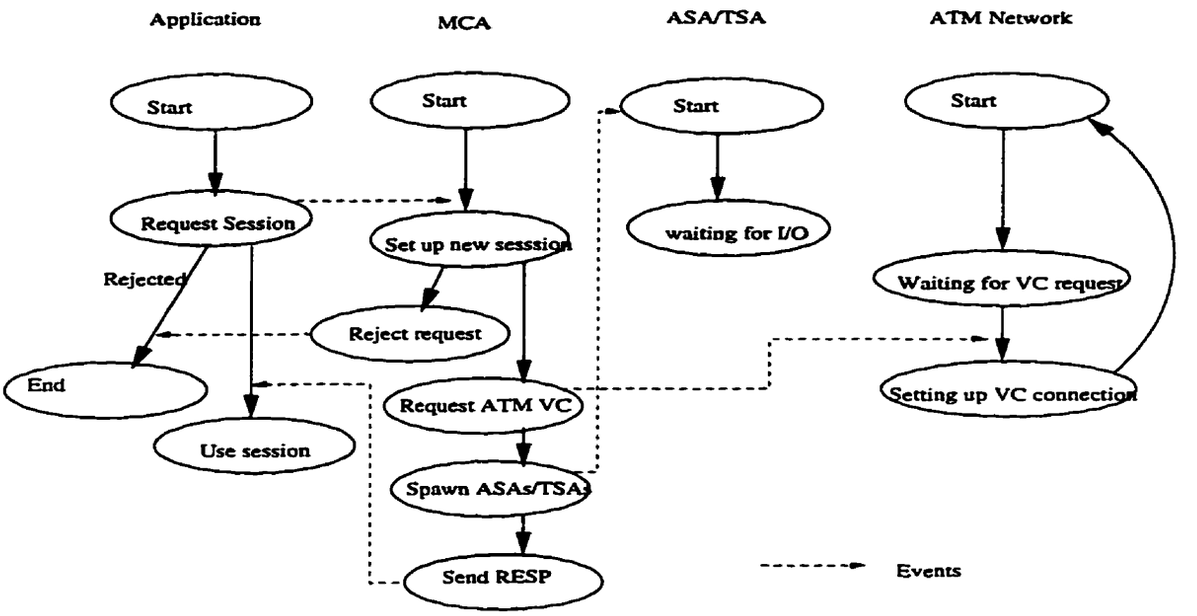

Figure 4.2: States in session establishment

transfer and video-on-demand (VOD).

The advantage of QRTP is that such a tradeoff can be configured for different applications because the length of queues between TSAs and ASAs is adjustable.

4.4 Connection Establishment and Termination

In previous subsections we have discussed the sequence of session establishment when discussing the behavior of entity MCA. In this subsection we present an illustration of states (Figure 4.2).

4.5 Resequencing

Due to the reason described previously, we have to assume TPDUs may arrive out-of-sequence. Inside the RTP header, we have the sequence number needed to correct such an occurrence. Algorithms for checking for out-of-sequence TPDUs are given in Appendix A.1 of the RTP specification [IETF-AVT95]. The specified algorithm, however, does not attempt to correct any out-of-sequence TPDUs so we present an algorithm for both detection and correction. The fact that we store TPDUs in a flexible receiving queue makes the task easier. The algorithm for sequence detection is illustrated in Figure 4.3. The algorithm for resequencing is in Figure 4.4. We assume `TPDUQ` is the queue of current TPDUs, `currentPDU` is a pointer to TPDUs in the queue, `GAPQueue` is the list of pointers to the TPDUs with a sequence number "gap" in front of it, and `DelayedQueue` is the list of pointers to TPDUs whose sequence number is smaller than the maximum of sequence numbers of all TPDUs in front of it. As an example, illustrated in Figure 4.5, suppose we have a sequence of TPDUs, with sequence numbers (12, 13, 14, 15, ..., 19, 20) when they are sent by sender T-TSA. When they are received by R-TSA the sequence numbers are (12, 14, 15, 17, 19, 13, 18, 16, 20). After running the detection algorithm, `GAPQueue` = (14, 17, 19, 20), `DelayedQueue`= (13, 18, 16). After running the correction algorithm, the corrected portion of TPDU queue is (13, 14, 16, 17, 18, 19, 20), so the new TPDU queue is (12, 13, 14, 15, 16, 17, 18, 19, 20).

Although the algorithm is important, its performance impact can be a concern for two reasons: (1) in order to correct the out-of-sequence problem the TPDU queue has to be locked while correction is taking place, and (2) if the algorithm is run

```
sequenceDetection(input/output TPDUQ, output GAPQueue, output DelayedQueue)
{
 initialize GAPQueue and DelayedQueue;
 previous = head of TPDUQ;
 current = second of TPDUQ;
 maxseq = current->seq;
 while (current is not end of TPDUQ)
 {
  if ( current->seq > 1 + previous->seq)
  {
     delete current from TPDU;
    add current into GAPQueue;
    maxseq = current->seq;
  }
  else if (current->seq < maxseq)
  {
    delete current from TPDU;
    add current into DelayedQueue;
  }
  current=current->next;
  previous=previous->next;
 }
}
```

Figure 4.3: Sequence Detection Algorithm

```
sequenceCorrection(input GAPQueue, input DelayedQueue, input/output TPDUQ)
{
 current = head of DelayedQueue;
 while (current != NULL)
 {
  currentGAP=head of GAPQueue;
  while (currentGAP!=NULL)
  {
   if (currentGAP->seq=current->seq+1)
   {
     insert (*current)  in TPDUQ;
     insert (*currentGAP) in TPDUQ;
     remove currentGAP from GAPQueue;
     remove current from GAPQueue;
     break;
   }
  else
  {
     currentGAP = currentGAP->next;
   }
  }
  current=current->next;
}
```

Figure 4.4: Sequence Correction Algorithm

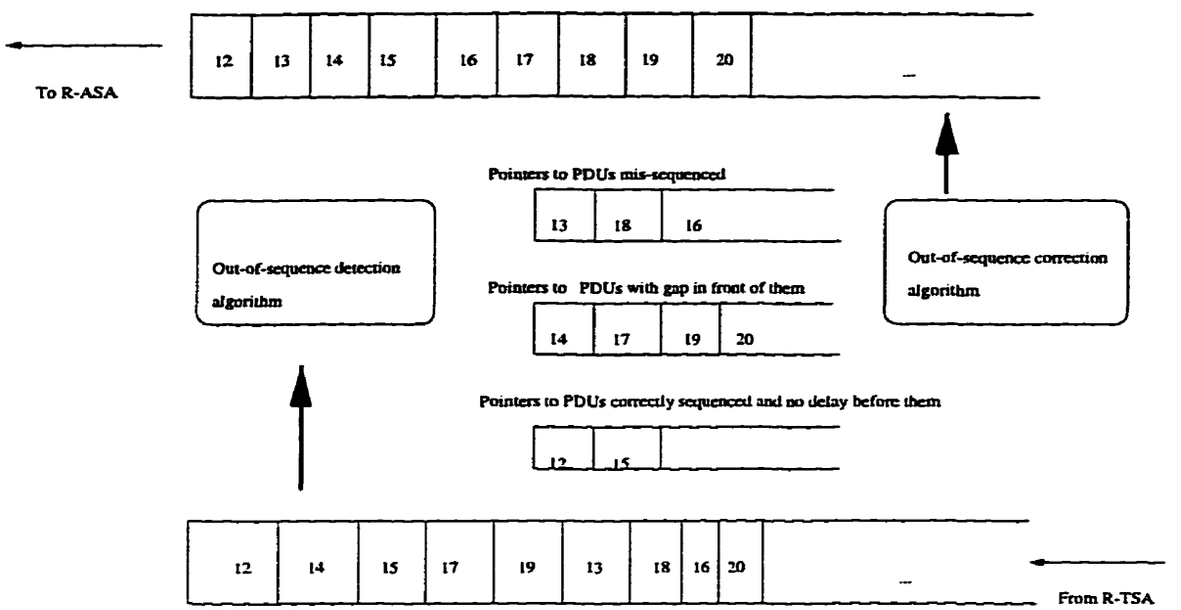

Figure 4.5: Algorithm for TPDU resequencing

too frequently it takes too large a percentage of processing time. Both issues can be addressed with a longer detection and correction interval, if the rate of out-of-sequence packets is small.

4.6 Synchronization of Different Sessions

Synchronization between different sessions is important in real-time applications. For example, University of Ottawa researchers identified that a 120 ms mismatch in lip-synchronization between an audio and video stream will be perceived as disturbing by users of a multimedia news service [LLG95]. It is therefore important to identify a synchronization algorithm for QRTP. However, they are mostly concerned about applications in which multimedia data is stored on a media server and users fetch data and play it back in a synchronized manner. Our research is concerned with

transmitting and playing back live data (such as video conferencing). The difference is that, while synchronization information (called a *synchronization scenario* in University of Ottawa's Stream Synchronization Protocol [LLG95]) can be defined for each multimedia data file beforehand in a multimedia server, it is impossible to store synchronization information for live data that has not yet been generated. As a result, synchronization information is specified between sessions at the transport layer.

Our design makes adding a synchronization algorithm easy. The process is illustrated in Figure 4.6. The synchronization process works as follows. After both sessions, say, A and B, have been established, a request for synchronizing them, *Session_Sync(A,B)*, is sent to the local MCA (sending side). The local MCA sends another request, *Session_Sync(A,B)*, to the remote MCA (receiving side). The remote MCA adds B to the list `sync_session_list` in the data structure of session A, and A to the list `sync_session_list` in the data structure of session B.

When the data is being transferred over both sessions, the TPDUs are timestamped at the T-ASAs of both A and B. We assume this algorithm handles only sessions from the same source, so these timestamps can be used to synchronize session A and B. At the receiving side, before dequeuing TPDUs of session A, the R-ASA of session A compares the timestamp of the head of session A's receiving queue, T_a, with the timestamp of the head of session B receiving queue, T_b. If T_a is at least ϵ ms greater than T_b, where ϵ ms is the *tolerable skew*, then session A is ahead and service of session A is stopped. At the same time, R-ASA of session B will do the same comparison, and it will find session B is behind session A, so it will keep serving

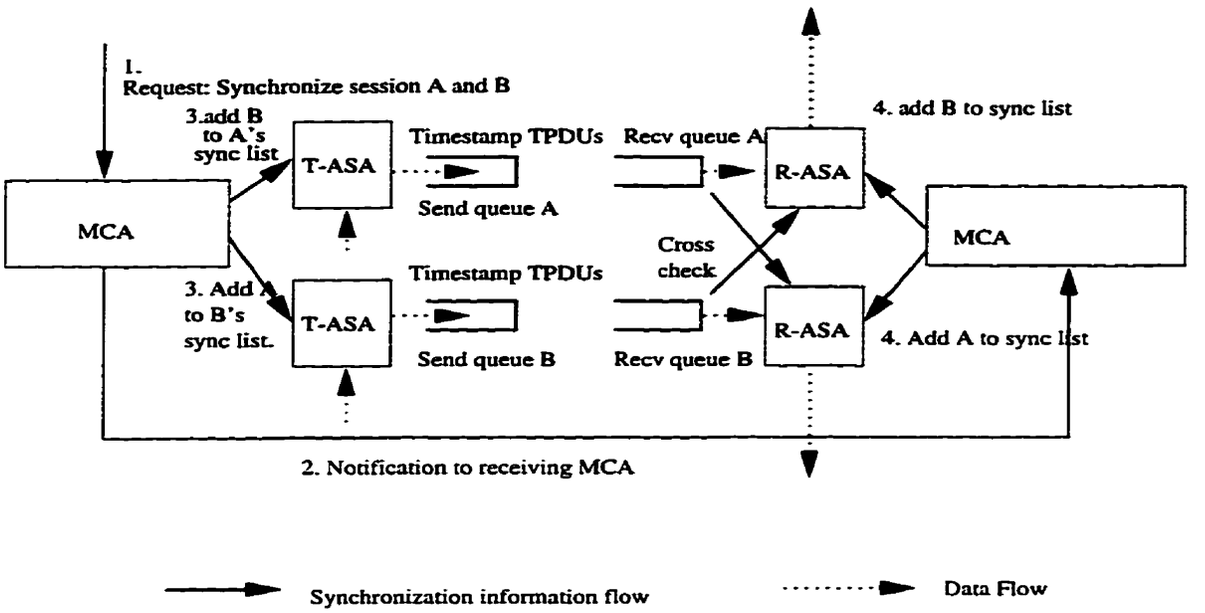

Figure 4.6: Synchronization of QRTP Sessions

until B is synchronized or ahead of A. With this algorithm, the maximum possible skew will be the maximum sending delay at the sending side.

This design has following advantages: (1) no predefinition of synchronization is needed, (2) configuration of synchronization is dynamic because it can be specified while sessions are working, (3) no extra synchronization protocol or controller is needed because we utilize timestamp information and ASAs.

4.7 Multicasting

We can approach multicasting in two ways. One is using ATM layer point-to-multipoint virtual circuits, and the other is maintaining a list of participants for every session and sending copies of the same data to all participants at QRTP layer.

Both app ... er case, a single virt
circuit id ... rtual c... cuit, is asso
ated with ... is appr... ach, of cour
is that th ... In the latter case,
ATM lay ... ipants are added in
the parti ... ferent ATM layer v
tual circu ... or mu... ple unicasti
operation ... ree (V... for transpo
multicasti

Multi ... ne in... tion comm
nication b ... serve... multicasti
server, w ... ... ca... ting) to all use
who are o

4.8 F

Error cont ... mission is an impo
tant part ... perf... s error contr
more effici ... mplementation we d
not perfo

Both approaches are easy to implement in QRTP. In the former case, a single virtual circuit identification, which identifies a point-to-multipoint virtual circuit, is associated with all participants of the session. The assumption of this approach, of course, is that the switching equipments support multicasting circuits. In the latter case, no ATM layer multicasting capacity is required. Multiple participants are added into the participants list of the session and for each there is a different ATM layer virtual circuit. A Multicasting operation is simply a combination of multiple unicasting operations. We are currently using ATM-layer multicasting tree (VC) for transport multicasting.

Multipoint-to-multipoint multicasting is made possible by the interaction communication between session owner and users. Owner, in this case, serves as *multicasting server*, which relays the information (specifically marked as multicasting) to all users who are on the user list of the session.

4.8 Error Control

Error control, including checksum, acknowledgment and retransmission, is an important part of TCP/IP. Since we use native ATM stack, AAL5 performs error control more efficiently with on-board firmware. Therefore in current implementation we do not perform error control at the transport layer.

4.9 Quality of Services Control

4.9.1 Specification

Transport layer QoS specification contains a fixed set of parameters. Compared to the application layer, a transport layer protocol provides limited types of services, therefore definition of their qualities is more limited. QoS is defined for each session, and contains the following attributes. Each of the attributes have parameters to represent its characteristics: maximum, minimum, average, and standard deviation.

1. End-to-end delay D. Suppose at time T_1 a transport sending entity emits a TPDU j, and at time T_2 the receiving entity accepts this TPDU then $D(j)$ is $T_2 - T_1$.

2. End-to-end delay jitter J. For a set of TPDUs indexed from 1 to j, J(j) is defined as $Max(D(j)) - Min(D(j))$. Here D(j) is the set of delay values suffered by all TPDUs.

3. TPDU processing delay t at both sending and receiving ends. Suppose T_0 is the time at which transport sending entity accepts TPDU j from an application entity, and T_3 is the time the receiving entity at the receiving party sends this TPDU j to application entity, and T_1 and T_2 are as defined in D_j definition, then $t_j = (T_3 - T_0) - (T_2 - T_1)$. That is, it is the different between application end-to-end delay and transport layer end-to-end delay.

4. TPDU error rate e. Suppose P TPDUs are sent out, and C TPDUs are correctly received by applications, then $e_i = (P - C)/P$.

5. TPDU generation rate, R. This is the rate at which TPDUs are emitted by the sending application entity.

6. TPDU processing rate, S. This is the rate at which TPDUs are processed by the receiving entity.

7. TPDU size L, in units of bytes.

4.9.2 Passing Requirements to ATM

The ATM Forum defined QoS guidelines for the ATM layer in its User-network Interface Specification (Version 3.1) (UNI 3.1) [ATMFORUM-UNI96]. UNI 3.1 QoS specifies both QoS parameters which should be common among ATM vendors, but also defines four classes of services and four levels of QoS corresponding to them. These service classes are: Class A, circuit emulation, constant bit rate video; Class B, variable bit rate audio and video; Class C, Connection-Oriented Data Transfer; Class D, connectionless data transfer. As defined by the ATM Forum, packetized video and audio in teleconferencing and multi-media applications will use Class 2 service. Based on this specification, any virtual circuit (VC) established by an MCA, for this type of applications will have both forward QoS and backward QoS set as class 2.

Once we have determined the transport level QoS for a particular application, we need to translate the values derived for parameters described in the last section into the values for Information Elements (IEs) of an ATM **setup** message. ATM Forum defines a set of QoS parameters for ATM layer, which includes cell transfer delay (average and variations), loss ratio and error ratio, etc. These parameters are monitored by measurement points in an ATM network. When they degrade, the internal

network implementation should address factors related to the degrading. However, in UNI 3.1 there is not yet a clear mapping between QoS of the bearer service and the network performance of connection elements supporting this service. That is, the monitored performance parameters are not comparable with bearer specification and traffic specification by signaling users.

How this specification should be done is illustrated in Figure 4.7. Some information elements are not critical but retained for completeness.

4.9.3 Enforcement and Policy: Resource Management and Scheduling

With the QoS information model defined, we next look at the control procedure that will convey the QoS information to the system and networks, in order to obtain the services satisfying the requirements. Control activities have to be carried out in *signaling stage* or *operational stage*. In the signaling stage, an application process on one end establishes transport sessions with other ends, presents end-to-end QoS requirements to lower layers, and reserves resources at lower layers. In the operational stage, lower layer control entities, such as the MCA at the transport layer and the network management entity at the ATM layer, monitor the QoS parameters and assert certain control actions when QoS degrades.

Table 1: QRTP SPecification of ATM Layer QoS

ATM QoS Information ELements	Explanation	Values
Protocol Discriminator	Determined by the type of signaling protocol chosen by the device vendor.	0=Q.931; 1 = Q93.B
Message Type	Type of the control message.	00000101 (=SETUP)
Message Length	Length of control message.	Default
AAL Parameters	Includes information such as AAL type to use.	AAL Type = 5.
ATM Traffic Descriptor	Includes peak rate specification.	Peak rate = (QRTP TPDU generation rate)/ 48.
Broadband Bearer Capability	Specify the characteristics of traffic service (CBR, ABR or VBR).	Traffic Type = 001 (constant bit rate).
Address Information for Called and Calling Parties	These information specifies how calling and called party can be reached.	
Connection Identifier	Virtual Circuit Identifier (VCI) and Cirtual Path Identifier (VPI) requested.	Variable
Quality of Service Parameter	Quality of Service level.	QoS = 00000010 (= Class 2).

Figure 4.7: Specification of ATM QoS by QRTP

Signaling Stage: Establishment

As we described in Section 4.9.1, application QoS can be broken down into different QoS subsets, each associated with one type of media. Complete QoS information for a application can be easily specified in terms of QoS attributes for each media object, then internally converted into transport layer QoS, represented by the fixed QoS parameters (D, J, t, e, R, S, L).

In a real-time multimedia system, it would be important for applications to check the availability of local system resources before signaling the network connections. Once it is confirmed that local resources are available, the QRTP API function `sessionInit` is called, with transport layer QoS parameters passed as arguments. This generates a request message to the local MCA, which first checks the availability of local transport resources, based on the defined mapping between local transport resource requirement and QoS parameters. If local resources are sufficient, it then passes the request to the remote MCA, which performs the same local transport resource confirmation and sends back a response to either admit or deny session establishment. If the response is denial, then session creation fails. If it is an admission, then local MCA translates part of the transport level QoS parameters to ATM layer QoS requirements, which are embedded in ATM **SETUP** message, then sends it to the ATM network. ATM Call Admission Control will handle the VC establishment and ATM resource reservation, then pass back result. If the setup is admitted, then a VCI is passed back for use. Otherwise denial is sent back.

At this point the establishment process is finished and the transport service for

the session is then handled by a newly spawned TSA, which is given the assigned VCI. The VCI is also part of the arguments the TSA will pass to ATM API calls such as Fore System's `atm_send()` to send data via the established VC.

Operational Stage: Monitoring and Control

Quality of service control by QRTP is a straight forward *credit-based* mechanism. The objective of the control is to distribute the endpoint processing power according to the QoS level assigned to each session, or amount of bandwidth guaranteed to each session.

At each endpoint system, there is a semaphore which is global within the endpoint transport system. Before any service agents (ASAs and TSAs) serve its session it has to obtain the lock. That is, at any point there is only one R-ASA is operating. By using this semaphore we schedule the operations of agents serving different sessions.

Before a session attempts to obtain the semaphore, it has to have at least one *credit*. Credits are assigned by the MCA according to the session initialization requests. There are two ways of assigning credits, according to two slightly different criteria. The first criteria is QoS level and the second one is bandwidth. When a session has credit, and it has obtained the semaphore, it can serve as many TPDUs as it has the number of credits, before it has to release the semaphore. That means, within a fixed amount of time, a session will occupy an amount of processing time proportional to its requested bandwidth or QoS level.

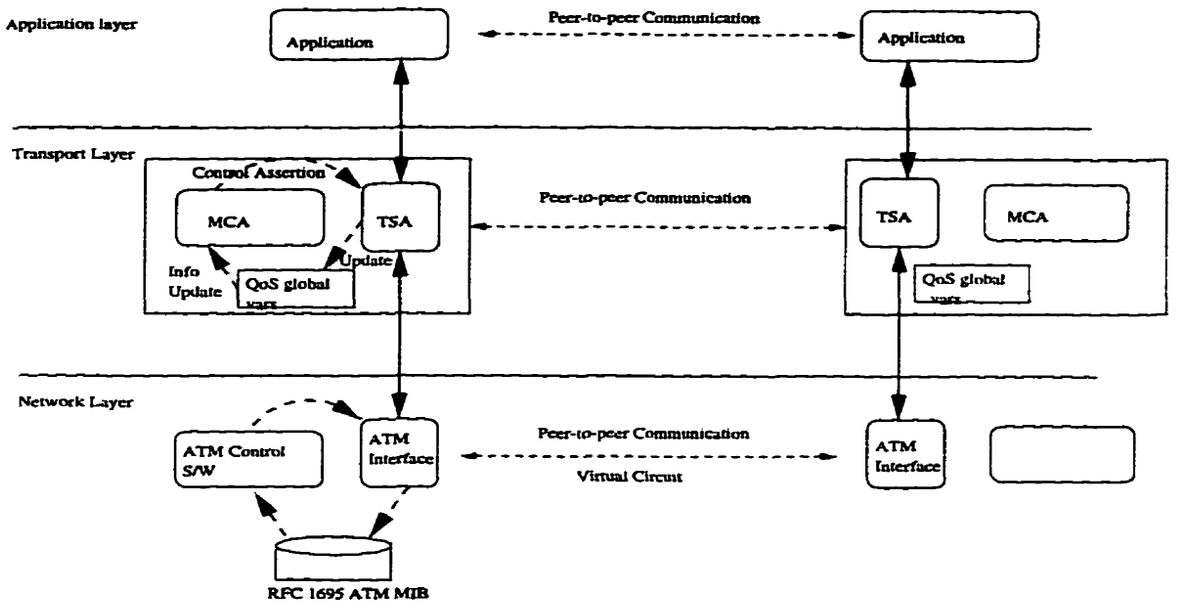

Figure 4.8: QoS Operational Monitoring and Control Procedure

Monitoring and measurement of QoS at the transport layer is done internally within the MCA, and at ATM layer by the ATM controller software usually residing on a control card of ATM switch, or the adaptation card. Another approach is to design a separate SNMP network management agent that obtains QoS information from a standard MIB (Management Information Base) [ATMFORUM-UNI96] [AT96] defined for each ATM virtual circuit or virtual path. An ATM MIB provides a rich set of performance information from which a complete picture of operational QoS can be derived. Work that needs to be done that is beyond this thesis includes a design of a mapping between this set of information and user's requirements, then onto the actions that have to be taken when user's requirements are not met. This is, however, within the scope of ATM design. The monitoring and control procedure is illustrated in Figure 4.8.

4.9.4 Example

Here we apply the proposed QoS specification and information model to an example. Suppose for a packetized video on demand application, the video server X should deliver 30 movie frames per second to client Y. Each frame consists of 1024x768 8-bit color pixels. Here we assume the worst case that no compression is applied (for example, if MPEG-1 is used, all frames are base B-frames). This means the application bit generation is 188,743,680 bits. Other application level QoS requirements are: error rate less than 6 frames per second; end-to-end delay should be constant; no frame should arrive at the client out-of-sequence.

At the transport layer, we select TPDU size L as 1024 bytes. Based on this information, D_i should be assigned a value based on distance and network topology between client and server. J_i is less than 1.7 microsecond. This means if frame A and frame B are both sent out within one second, and frame A suffers the longest delay, and frame B takes the shortest delay, they should still be at most 1 second apart, otherwise arriving rate for that second can be less than 30 frames. So approximately for every RTP TPDU which has size 1024 bytes, jitter J_i tolerated for PDUs is 1.7 microsecond. Since 23040 TPDUs have to be processed by transport layer, each TPDU can only take t_i 43 microseconds. Since maximum error rate is 20%, so e_i is 20%. R is 23040 TPDUs, and S is also 23040 TPDUs.

At ATM layer, ATM forward and backward QOS are defined as Class 2. Forward and backward maximum SDU values are set to 8192 bits. All peak rate values are set to be 3,932,160 ATM cells per second. Since this VOD application is an end-to-end

application we set User Plane Connection Configuration to be 01.

QoS parameters derived above are passed to various control entities, including MCA and ATM network for signaling.

4.9.5 Session Scheduling

Session scheduling is an important method to ensure that TPDUs from a session for which higher QoS is specified is devoted more endpoint CPU power (hence transport bandwidth). We base our design on two basic mechanisms: (1) a locking mechanism which uses a unique endpoint mutex lock, and (2) a credit-based TPDU service model in R-ASA. See Figure 4.9 for an example on how this scheme operates.

For each R-ASA, before it serves the TDPUs in the its incoming TPDU queue, it locks a mutex lock, which is global within the QRTP endpoint. This makes sure there is at most one R-ASA utilizing the CPU processing time that QRTP has. Then it loops, serving one TPDU in each loop. The critical part is that, the number of times the R-ASA loops before it gives up the lock, is determined by the level of QoS it requests. For example, a session with QoS level 5 can serve 5 TPDUs before it contends for the lock again, while a session with QoS level 1 can only serve 1 TPDU every round. When the lock is free, it is determined randomly, by the operating system who gets the lock next.

This algorithm is simple and simulation shows that it indeed yields more processing power to the sessions with high QoS level.

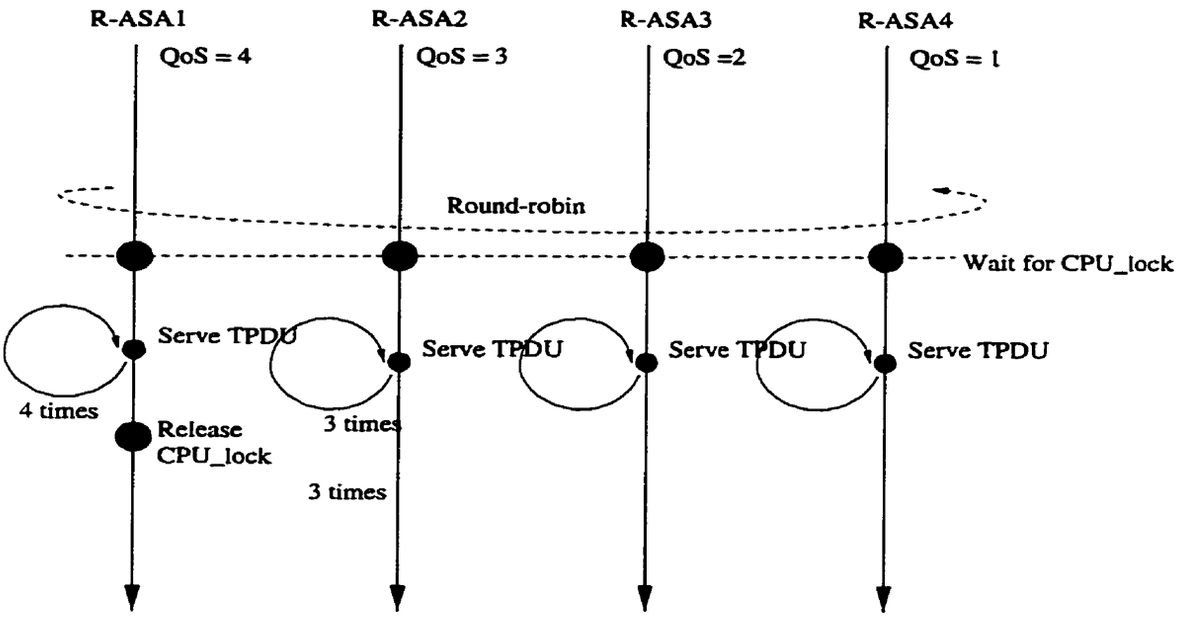

Figure 4.9: QoS Scheduling

Chapter 5

QRTP Implementation

Implementation of QRTP is done in the C language, based on the assumptions that (1) a thread library is supported by the operating system, and (2) an ATM API is provided for programming. In this chapter we will take a rather high-level view of the implementation of modules and examine some implementation issues. In Section 5.1 we give an overview of the implemented system. In Section 5.2 we introduce the main data structures used in the implementation. The modules are described in Section 5.3. In Section 5.4 some implementation issues are described.

5.1 Overview

There are three types of modules in the QRTP implementation, as illustrated in Figure 5.1. They are:

1. *Interface library modules.* These are the modules which implement the client API used to access QRTP services, and handle the communication between application and protocol entities. These modules are archived into a single

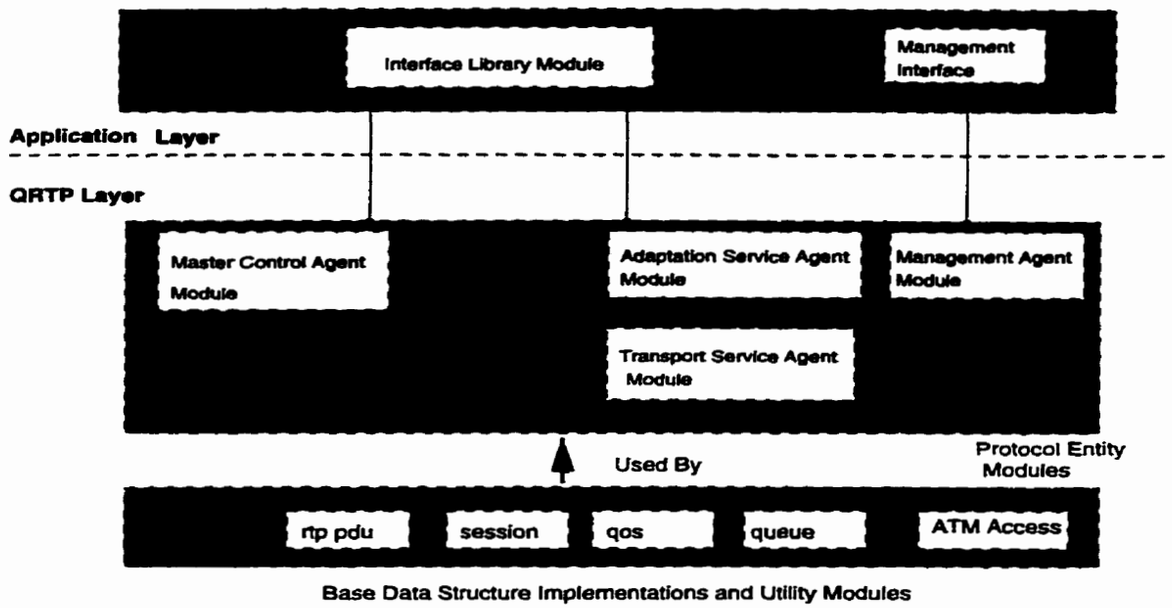

Figure 5.1: Main modules

library, `libqrtp.a`, which can be linked with application code.

2. *Protocol entity modules.* These are modules performing the tasks of the MCA, ASAs, TSAs and MAs. They run as separate process or threads and communicate among themselves and with applications. These modules are linked into a single executable, **qrtpd**.

3. *Base data structure and utility modules.* These are data structures (such as session) and their associated manipulation functions (such as session initialization), or utility data structures and manipulation functions (such as queue and queue manipulation).

5.2 Data Structures

The main data structures are

- `RtpProtocolDataUnit`. This is a data structure implementing the RTP V2 TPDU format. It contains the standard RTP header, an extension header, and a fixed length payload. In the extension header, extra real-time control information has been entered to suit our needs (see Figure 3.1).

- `RTPSession`. This data structure contains attributes of a session, and a Protocol Control Block which contains performance-related information. Since there is a one-to-one mapping between an existing session and an ATM VC, there is also an attribute holding the ATM VPI/VCI pair which represents the ATM VC that supports the connection.

- `RTPSessionList`. This data structure represents the list of sessions.

- `QoSTransport`. This data structure represents the QoS information of each session.

Along with these data structures, there are manipulation functions associated with them.

One data structure which needs elaboration is `RTPSessionList`, which is a list of `RTPSession` data structures. The list, which is shown in Figure 5.2 is a global data structure within an MCA, and it is accessible by all service agents. Attached to every `RTPSession`, there is not only session information, but also buffer queues of TPDUs to send and TPDUs received, and other operational information that change when the session is in operation. That is, `RTPSession` is not only used as an information repository of a session, but also an operational buffer of the session.

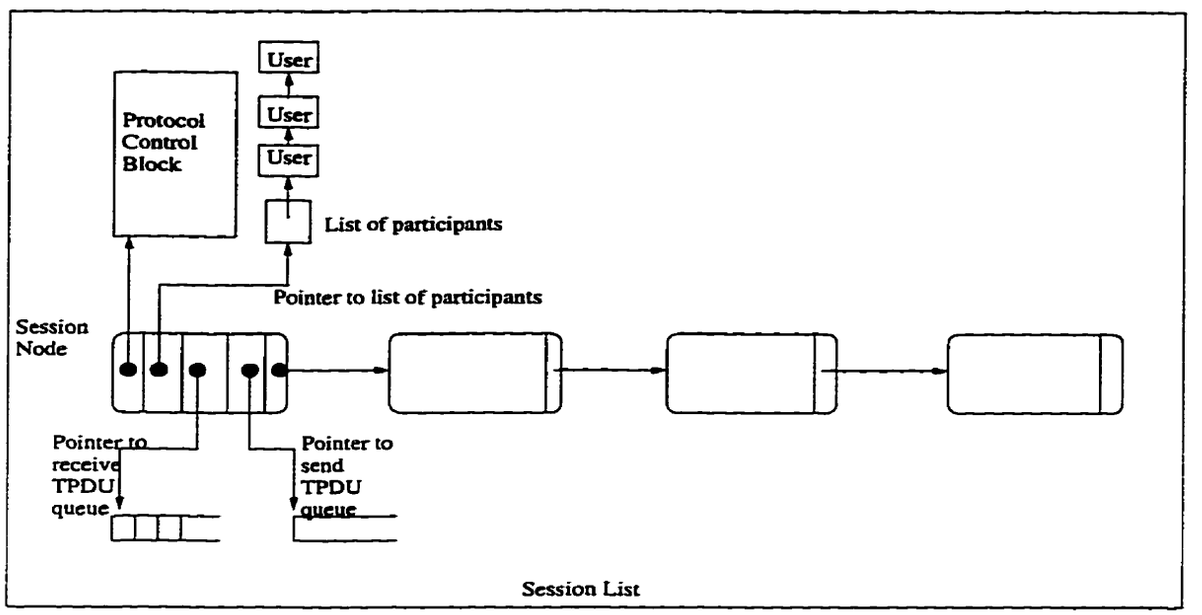

Figure 5.2: Data structures

The Protocol Control Block (PCB) of a session is part of the session data structure. It contains attribute sets that are updated by different protocol entities. These sets are all part of the session global data structure but do not overlap so they can be updated without any locking mechanism. PCB contains the following information sets (see Appendix C):

1. Information updated by the MCA which includes `sessionStartUpTime` and `recvWin` (receiving window size).

2. Information updated by T-ASA agents which includes the queuing time of the last TPDU onto the sending queue.

3. Information updated by T-TSA agents which includes

   - `totalSentPdus`, total number of TPDUs sent;
   - `currentTotalSendQueueDelay`, total queuing delay so far.

4. Information updated by R-TSA agent, which includes:

   - `currentWinExpectedPdus`, total number of TPDUs expected. This is used to check TPDU loss;
   - `currentWindowActPdus`, total number of TPDUs received in the current window;
   - `totalExpectedPdus`, total expected TPDUs;
   - `totalRecvPdus`, total received TPDUs;
   - `totalLossPdus`, total lost TPDUs.

5. Information updated by R-ASA agents, which includes:

   - `currentTotalRecvQueueDelay`, total queuing delay by all TPDUs so far;
   - `lastPduTime`, the time when last TPDU is processed by R-ASA, which is used to calculate processing rate and inter-arrival rate;
   - `currentInterPDUInterval`, current arrival interval;
   - `lastRecvPDUSeq`, last received TPDU sequence number;
   - `totalRecvOutOfSeq`, total out-of-sequence TPDU numbers.

5.3 Modules

5.3.1 Application Interface Functions

Interface functions are used by applications to access the services offered by QRTP. Services are available via message exchange, and interface functions encapsulate the sending, receiving and processing of the messages.

Functions: qrtpInit and qrtpRemove

Function `qrtpInit` takes a pair [session initiator host address, session id] as one parameter. The other parameter is QoS level specified for the new session. Its function is either to (1) initialize a new session, or to (2) join the session identified. It first establishes a connection with the local MCA, then sends a message of type SESSION_INIT (if case (1)) or SESSION_REMOTE_JOIN (if case (2)) to the local MCA. It then listens for a response back from the MCA. If for some reason the session cannot be set up, the MCA responds with a negative response and the application is notified. If the response is positive, then it contains the contact information (such as a socket id) of the ASA agents that will serve the session from this point on. This information is stored in the application and is used to access the session. UNIX sockets are used to communicate with the local MCA.

Function `qrtpRemove` sends messages to the local MCA to terminate a session. As well, it clears up all the internal data structures associated with the session,

Function: qrtpSend

After an application establishes a session, it sends data over the session by calling this function. The function simply passes the data to the T-ASA agent via a UNIX socket.

Function: qrtpRecv

After an application establishes a session, it receives data over the session by calling this function. The function simply passes the data from the R-ASA agent via a UNIX

socket.

Function: qrtpSync

This function is called to synchronize two sessions. Best effort is made to ensure that, when two sessions are synchronized, if the sending times of TPDUs from two different sessions are within a certain range of each other, they will be served at the receiver end within the time interval as well.

Function: qrtpWithdraw

This function is called by an application entity to withdraw from a session.

5.3.2 Agent Functions

Functions and internal logical flow of the MCA, ASAs, TSAs, and MA were described in Section 3.3. They are implemented as a separate layer on a UNIX host. The MCA is unique in a single endpoint. It is implemented as a UNIX process (a main function) and performs control functions only. It listens on a known port and communicates with local applications via a UNIX socket. It runs in an endless loop, listening for requests from local applications for remote MCAs. The MCA employs a *concurrent* service model, which means it retrieves a request, forks off a thread to serve it, then listens for the next one. It is the parent process of the service and management agents (TSAs, ASAs, MA), which are each implemented as UNIX Solaris threads. These agents are started within an MCA by calling `thr_create` and have access to any global data structures, including the session lists maintained by the MCA. Each session's service agents access the session information in memory. An MA obtains

management information and provides to management applications.

All agents work concurrently. There are *critical areas* where scheduling is needed. Access to sending queues and receiving queues of the same session by different agents is controlled by semaphores.

5.4 Implementation Issues

5.4.1 ATM Connection Establishment and Usage

We use the Fore ATM API to access Fore Sparc/Solaris ATM Network Interface Cards, which are connected via Newbridge 36150 ATM switches. Since at this stage the ATM switches we use do not support signaling, switched virtual circuits (SVCs) can not be set up dynamically. As a result, we have to use permanent virtual circuits (PVCs) to support sessions. When an MCA is setting up a session, it calls `atm_connect_pvc` to connect with an incoming PVC (identified by VCI/VPI pair). A file descriptor, which represents the ATM connection, is returned and stored in session information. After T-TSA and R-TSA for the session are started, they use the file descriptor as a parameter, to call `atm_send` or `atm_recv`, for sending and receiving data.

In the current implementation, mapping between session IDs and ATM VPI/VCI pairs is defined in a database file, which is loaded into the global memory of the MCA when it is started. The MCA then assigns VCI/VPI to every session when the session is established. Only one receiver can use one VC to receive data. The reason is, if multiple recipients were allowed to listen on the same ATM VC, any one of them might

get the data and no one else would get it, thus the wrong receiver might "intercept" the data from the correct one. However, multiple senders can share the same VC for sending data. In multimedia applications, receivers often have to identify the source of data. As a result, if multiple senders share the same VC, source definition is achieved with the use of the SSRC field RTP header. In this implementation, to simplify the current implementation, we assume there is a one-to-one mapping between a session and a VC.

5.4.2 Protocol As Part of Application Process vs. Separate Entities

Two options were available to us when implementing QRTP:

- Implementing QRTP as a library of function calls, which is linked with the application code into a single executable.
- Implementing QRTP as a totally separate set of entities, which communicate with application entities via a message-based interface, provided in form of a library.

The advantages of the first option are: (1) interaction between application and transport functions is as easy as function calls which makes it simple to pass information, (2) passing of control and payload information from application to transport services is within memory therefore fast. The advantages of the second option are: (1) all sessions on an endpoint will be under control of one single MCA, which is impossible under the first option, (2) global resource reservation and QoS control among sessions are easier, (3) adding new control functions is easier since we can design them

as separate threads which are started by MCA, and it will have access to all session control data, and (4) this is the model used by the Internet services, where Internet daemon `inetd` serves the Internet communication of all applications on a host. We selected the second option based on the advantages presented. Our selection presents good separation of functions and global control of the QRTP services on an endpoint. The main shortcomings, however, is that communication between QRTP users and QRTP daemon must be done via UNIX sockets instead of via memory as in the first option.

5.4.3 TSA only vs. Combination of ASA and TSA

Originally we attempted an implementation where there were no ASAs between the application entity and TSAs. Our experiments revealed that, while the sender side works fine, the receiver side is unable to process the received PDUs fast enough. As a result, a large number of TPDUs (30%) are lost. We looked at three solutions: (1) Increasing the number of TSAs so that we have a concurrent service model for *each* session, at the receiver's side, and there is always at least one TSA listening for incoming TPDUs when others are processing received TPDUs; (2) reducing the workload of the TSA to merely receiving PDUs and saving them in the buffer, and creating another entity, called an Adaptation Service Agent, to further process the TPDUs, convert them back into payload and send to applications; (3) perform flow control to slow the sender down. We concluded (3) is not desirable in a real-time situation. (1) is a better solution but experimentation showed that although PDU loss rate reduces when the number of TSAs increases, the loss rate does not reduce further after the number of TSAs reaches a threshold. Therefore (1) is not a good

answer, either. We implemented (2) and experiments showed that when the sender endpoint and the receiver endpoint processing power is equivalent, the maximum loss rate is virtually 0% and playback delay is small. We concluded that, with a combination of TSAs and ASAs, we can achieve low loss rate and small playback delay. Our experiments proved that with an extra a few 1/100 seconds of delay, which is negligible to human perception, we avoid at least 30% of TPDU loss which is extremely visible.

Chapter 6

Application Development Using QRTP

Figure 6.1 outlines the general structure of the example applications which use the QRTP API. In the following sections we describe some applications that we developed to evaluate QRTP.

6.1 QRTP Control Utility

To acquire both a testing application and a control interface, we implemented a QRTP controller, which has both a GUI and a command line based interface. The functions of the controller are (1) to issue control commands to the MCA, (2) to perform monitoring functions, which means users can issue queries to the management agent and get responses detailing the status of each session.

Figure 6.2 is a snapshot of the controller when we initialize a session. When we

```
#include "cbartp.h"
#include "cbartp_master_if.hxx"
#include "cbartp_access.hxx"

CBARTPSessionAccessInfo *session1, *session2;

int main()
{

 dataType   buffer;
 char*      otherHostName;
 int        otherSessionId;
...
  /* Set up a session that this application owns */
  session1 = qrtpInit(Session_Init, NULL, 0);
...
 /* Send data to other participants */
 qrtpSend(buffer, session1);
...
 /* Terminate session */
 qrtpRemove(session1);

/* Join a session that other application owns */
 session2 = qrtpInit(Session_Join, otherHostName, otherSessionId);
...
/* Receive data */
  qrtpRecv(buffer, session2);
...
 return 0;
}
```

Figure 6.1: Overview on how APIs are used

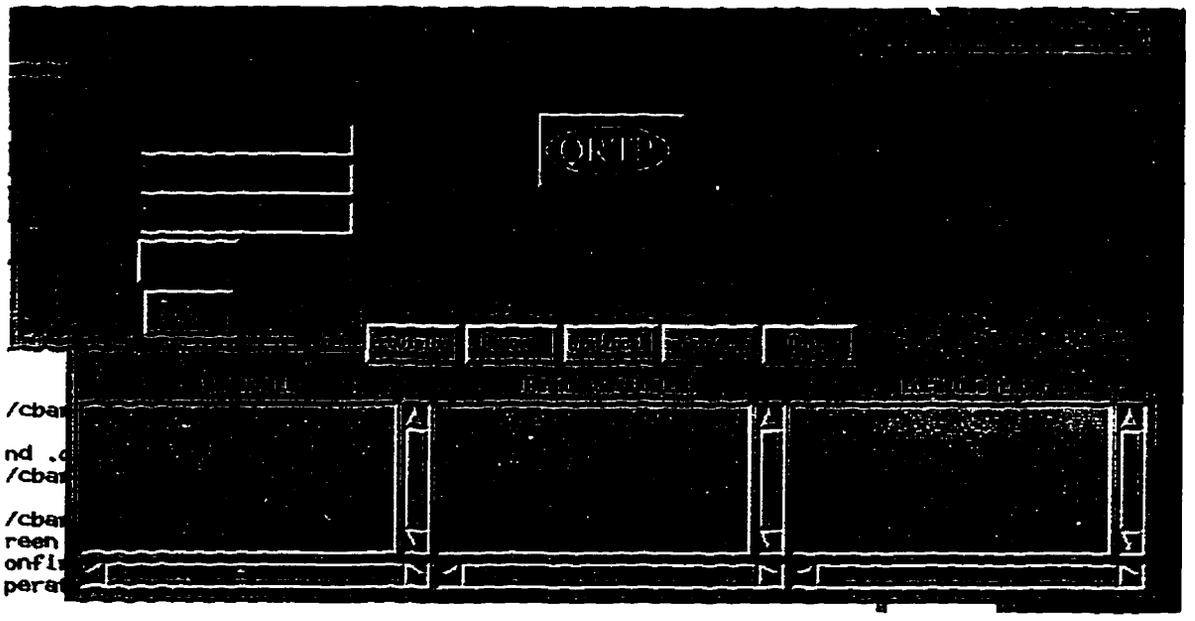

Figure 6.2: Tk-based QRTP Controller when provisioning a session

select `Provision`, the controller presents a dialog box, asking for session name, host name and session ID. Once the information is entered, we press `Commit` to send the `SESSION_INIT` request to the MCA. After the session is initialized, we can display it by selecting `List Local Sessions` which sends a query to the MA requesting the list of local sessions, and displays them at the leftmost box when the response arrives.

6.2 A Live Video Broadcasting Application

The video broadcasting application was ported from source code of an example shipped with Sun's XIL library. Its operation is rather simple:

1. An MCA is started on all hosts involved;

2. a video sender application entity (**videosend**) is started from a workstation where a camera is residing (the source of the video);

3. videosend establishes a new session with its local MCA;

4. a video receiving application entity, **videorecv**, is started on each of the other workstations;

5. each of the instances of **videorecv** joins the session established in Step (3), by contacting their local MCAs;

6. broadcast video images source host to all destinations.

This application tests the basic correctness of QRTP and demonstrates the following main features:

- Multicasting capability.
- Low loss rate (by running a monitoring application, **qrtpstat** at the same time to measure and calculate the loss).
- Ability to handle live, real-time data to the best extent of host processing power.
- Connection-oriented communication.

Video image capture and compression are done by using Solaris Xil library function calls. Both Cell-B and JPEG compression can be used. Video data blocks, captured using Xil functions, are sent using the function **qrtpSend** and received using the function **qrtpRecv**. In Figure 6.3 we set the display of the sender and multiple receivers onto the same screen so we can capture them in the same snapshot.

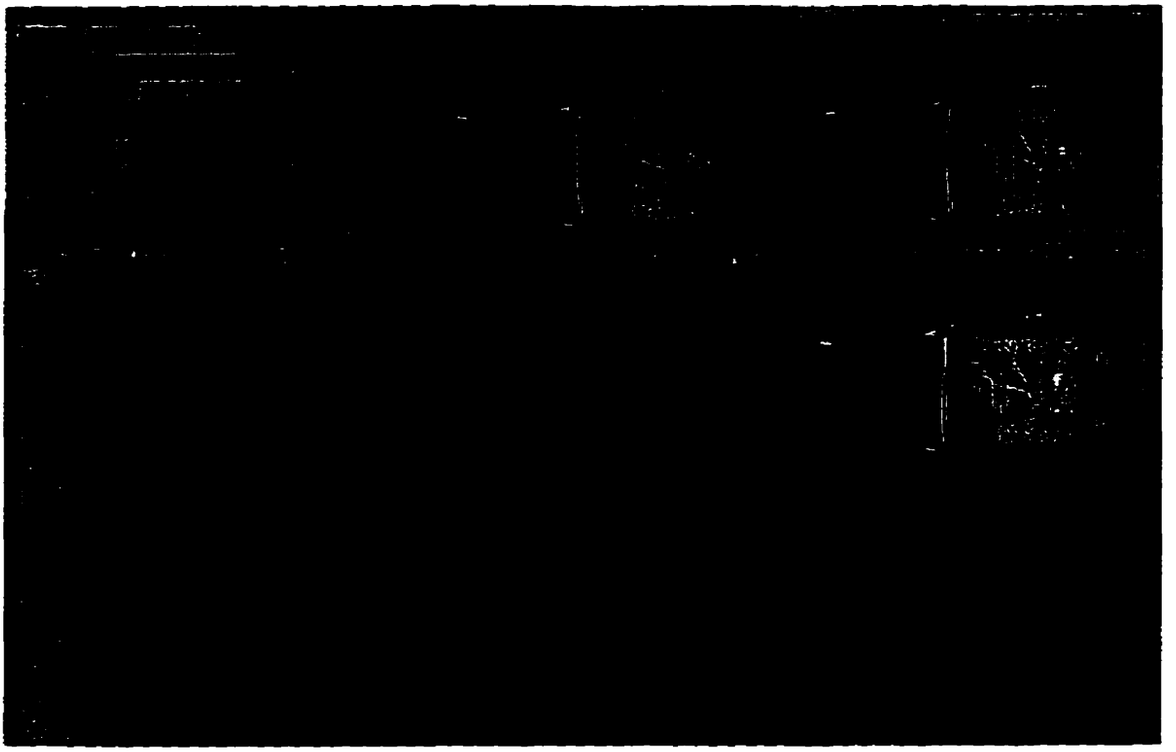

Figure 6.3: **Running video broadcasting from source to two destinations.** An interesting experiment is to visually observe playback delay of the clock image. If exactly 60 seconds elapsed at the sender side image (hand going exactly one circle), then the receiver side image should show the hand going exactly one circle in exactly 60 seconds. If it takes more than 60 seconds, then we may conclude significant delay is inserted into the transport process.

6.3 An Audio Broadcasting Application: `audio_multi`

The application `audio_multi` is used to test how QRTP can be used to perform packetized audio transport. Both the sender and the receiver run the same tool. The sender has access to a pre-recorded audio file, and it first initializes a session. Receivers then join the session. Once a user tells the sender to start, it fetches audio data from the file and sends it in real-time to all receivers. Receivers play back the data, also in real-time, on the audio systems.

6.4 An Application Testing Synchronization

In order to test how well the synchronization works, the application simultaneously sets up two sessions between two processes. The controller is used to synchronize two sessions and synthetic data is sent over both sessions. The departing time and arriving time, at the application level, are recorded by the sender and receiver, respectively. The same experiment is then performed without synchronization. Results from both experiments are compared. The conclusion is that the unsynchronized experiment yields larger inter-session gap, in terms of end-to-end delay.

Chapter 7

Performance Evaluation

The objective of the performance evaluation is to review how efficiently the current implementation provides services, and to discover the advantages and disadvantages compared to a conventional TCP/IP stack. One important part of the performance evaluation is to evaluate whether the use of the hybrid addressing scheme is justified. This requires an experiment to compare the performance of payload transport over IP/ATM and over ATM stack directly.

7.1 Queen's University Multimedia Networking ATM Testbed

The Queen's University Multimedia Networking ATM Testbed is illustrated in Figure 7.1. Two Newbridge 36150 ATM switches are connected via Queen's University backbone optical fiber network. Two Sparc 5 workstations are each connected to one

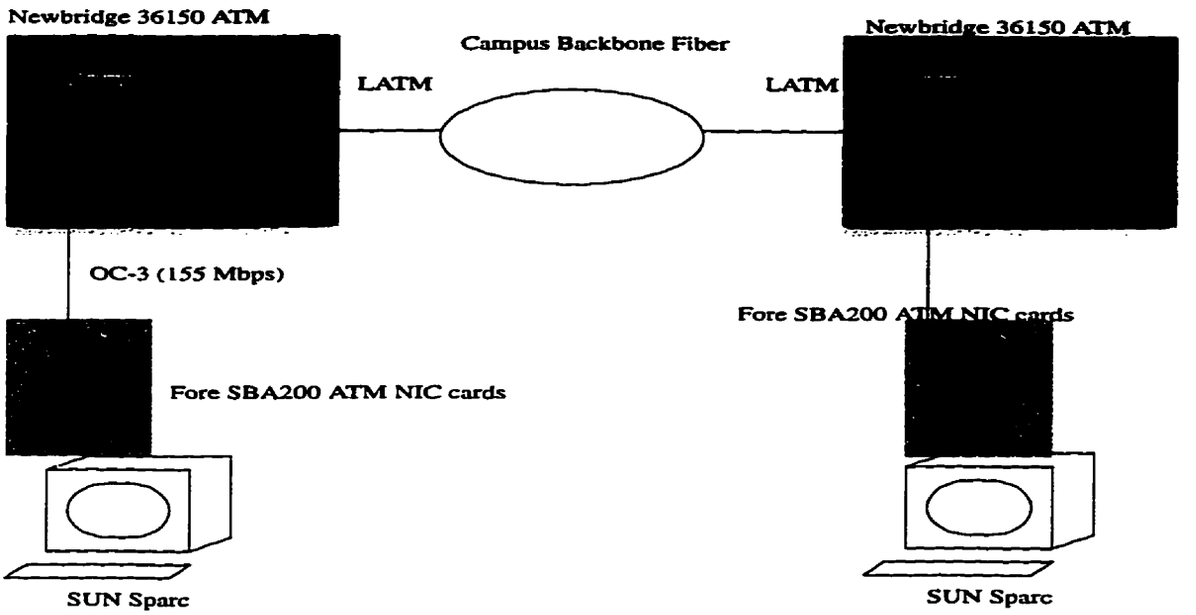

Figure 7.1: Queen's ATM Testbed

of the switches and each is equipped with multimedia devices such as a camera, a microphone and speakers. The bandwidth between workstations and the switches is 155 Mbps (SONET OC-3), while the connection between switches is 140 Mbps (LATM). At the workstation end, Fore System's SBA200 ATM network interface cards (NICs) are used to connect to the switches. There are two ways to access the switches: via standard TCP/IP stack or via native ATM stack. The first method is supported by an IP over ATM implementation and a standard socket interface. The second method is supported by an ATM API from Fore Systems. Switched virtual circuit (SVC) signaling is supported by the Fore NIC cards but not supported on Newbridge ATM switches. So, we must manually provision a VC and have the MCAs at both ends connected to it before proceeding with the testing.

7.2 General Method

The general methodology was to have controlled traffic generated from one endpoint, sent to a remote endpoint, and then fed back to the originating endpoint. The traffic samples collect information along the path and are used for calculating performance information. Traffic is simply sent out from sender as quickly as possible, and intervals are inserted to control the load of the system. Performance information is measured with one session running in the system performing both sending and receiving. Readers should be reminded that the traffic load was generally light on the current ATM testbed when data collection was performed.

Our performance measurement and analysis focuses on *latency*, *throughput*, and *loss rate*. Figure 7.2 depicts the *measurement points* (A, B, C, D, E, F) in the communication process at which we collect performance information. The information includes

- The time an application sends out a sample and the time it receives the same sample back. The time interval is used measure the user-to-user delay.
- Receiving queue waiting time (queuing delay), referred as D_{recvq} hereafter, is calculated from observed values at point D and E. It is used to measure the application transmitting rate.
- Arrival intervals of payloads from application sender to transmitting ASA agent (T-ASA), referred as $I_{sendarrival}$ hereafter, are collected from point A.
- Sending queue waiting time (queuing delay), referred as D_{sendq} hereafter, is calculated from observed values at point B and C.

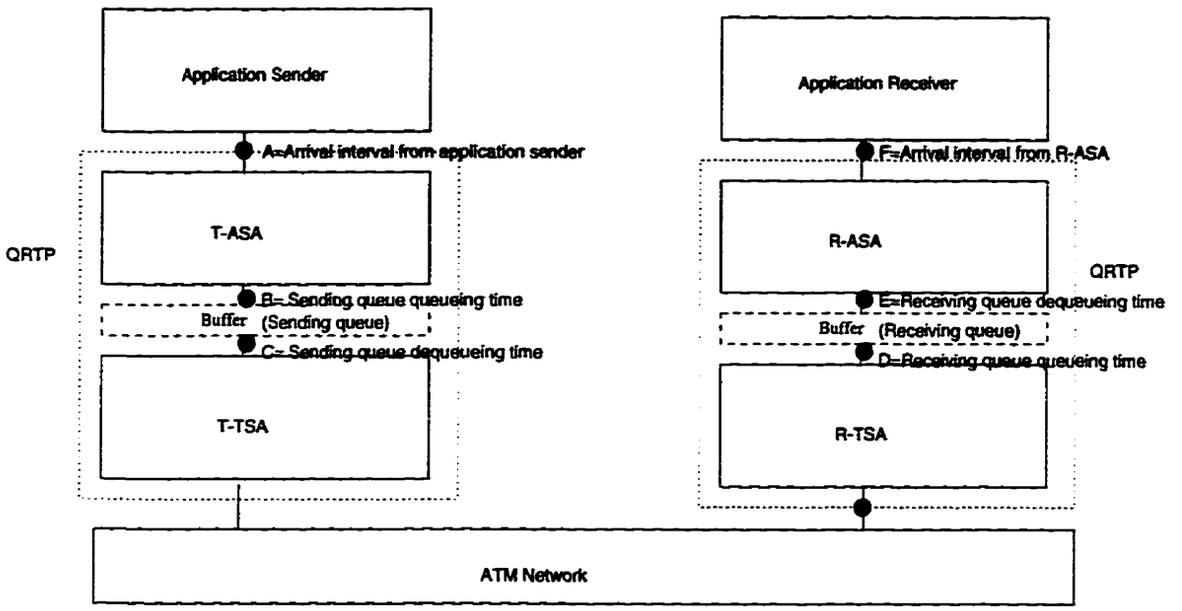

Figure 7.2: Performance Evaluation Information Collection Points

- Arrival intervals of payloads from R-ASA to application receiver, referred as $I_{recvarrival}$ hereafter, is collected at point F. It is used to measure the receiving rate, which is the effective throughput of the system.

7.3 Latency

7.3.1 Methods

Latency is the end-to-end delay between applications (referred as L hereafter). In our experiment, we measure the round-trip delay and estimate the end-to-end delay is half of the round-trip delay. There are extra values inserted into the estimated delay obtained with this method but the real delay will only be *better* than the estimated value. Each user data unit sample is marked with a sending timestamp when it is sent to the local QRTP T-ASA. The remote end sends it back immediately when it

is received. When it is received by the original sending application, the sample is marked with a receiving timestamp. Timestamp information of all samples are processed *after* the run has been finished, to calculate the means and standard deviations of samples of round trip delay. Since delay is related to the traffic load (sending rate) and throughput (processing rate), information regarding the relationships between latency and these metrics are also collected.

At the same time, we are interested in main elements of the latency. We can decompose D as follows:

$$L = D_{recvq} + D_{sendq} + D_{atm} + D_{trttrans} + D_{rrttrans} + D_{rasa} + D_{tasa}$$

where D_{atm} is the transmission delay in ATM network, and $D_{trttrans} + D_{rrttrans} + D_{rasa} + D_{tasa}$ is the sum of the service time at all four entities through which traffic passes. Experiments revealed that the sum of $D_{atm} + D_{trttrans} + D_{rrttrans} + D_{rasa} + D_{tasa}$ is on the order of microseconds, while D_{recvq} and D_{sendq} are on the order of $1/100$ seconds. Therefore the statistics behaviour of these two metrics interests us the most. When the QRTP daemon is started with the option of performing performance tracking, it logs the timestamp information of selected TPDUs, which is recorded at different measurement points. The log is kept in the memory because writing to disk would take extra time. The shortcoming of keeping the log in the memory is that the log has to be small so as not to affect performance of the daemon. This log is analyzed to extract the mean delay at different buffer queues. This helps us to draw conclusions on where the majority of the delay comes from.

For the round-trip delay experiments, sample size is determined according to a

method given by Jain [Jain91]. The method is described as following:

$$N = (100zs/rx)$$

where

- z is the normal quantile necessary for a two-sided confidence interval at some specified level of confidence;
- s is the standard deviation of the original sample;
- r is the desired accuracy level;
- x is the mean of the original sample; and
- n is the resultant necessary sample size.

We require an accuracy of 5% and confidence level of 95%. We select different traffic loads, under each of which round-trip delay information is collected. The number of samples range from 250 to 800, depending on the traffic loads we select, in order to ensure the required accuracy and confidence level under each load. For calculating means of queuing delays, sample size is determined in the same way. 747 sample TPDUs are needed to derive the mean of receiving queue delay and 366 sample TPDUs are needed to derive the mean of sending queue delay.

7.3.2 Analysis

The mean of round-trip delay is plotted against the traffic load in Figure 7.3. Under light load (44 TPDUs or 1.2 Mbps), the delay is about 164 ms. Ahuja, Keshav and Saran showed in their performance measurement of kStack [AKS96] that a 64-byte

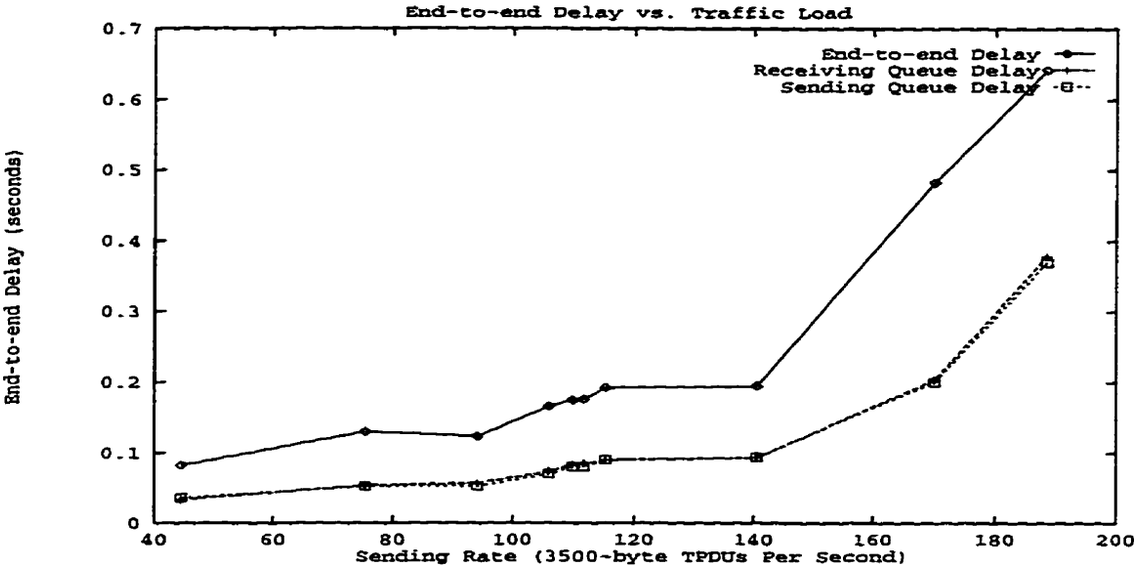

Figure 7.3: Round Trip Delay Under Different Traffic Loads

TPDU has 0.120 second round-trip delay. Under light load, round-trip delay of most of TPDUs (3000 bytes each) in QRTP is close to this value. Therefore, we have acceptable performance in terms of end-to-end delay at light load.

Under heavy load, since our protocol ensures low loss rate with a flexible size queue, and daemon processing power (throughput) is limited by the endpoint system, delay increases to a high level, mainly due to high queuing delays. For example, at 188 TPDUs per second (5.2 Mbps), throughput is limited at 159 TPDUs per second, and it results in a 1.2 second round-trip delay. There are ways to improve it: (1) moving QRTP into kernel address space, (2) using an array-style implementation of buffer rather than dynamic linked list style buffer, (3) using an interrupt-based synchronous service model to replace current queue-based asynchronous model.

7.4 Throughput

For throughput, we are concerned about the maximum processing power of both incoming and outgoing TPDUs, which are also measured by the experiments which is used to measure round trip delay. The advantage of using such experiments is that in such a "worst case" one QRTP daemon handles both incoming and outgoing sessions at the same time, under heavy load. The maximum throughput observed from iterations of such experiment is 161 TPDUs (4.5 Mbps).

7.5 Loss Rate

Each TPDU is numbered by sequence numbers, and they are logged when they are sent and received. The two logs are compared, after the experiment, to calculate the number of TPDUs lost. The number of experiments required to achieve a 95% confidence interval is 420, and it is determined with the method mentioned above. Each experiment sends out 2000 TPDUs (3500 bytes each) without interruption.

An average 2% of TPDUs are lost in the transport process. Such a small loss rate is a large improvement from TSA-only implementation. An earlier measurement on using TSAs only (no ASAs), for the same application, revealed a 30% loss rate. A R-TSA agent has to receive TPDUs, process them, then forward them to the application. From initial observations we concluded that this process takes a relatively long time and when other TPDUs arrive, there is no agent to receive them. To deal with this problem, we also implemented a version where multiplex TSA agents are started simultaneously to serve one session.

7.6 Comparison of QRTP over IP/ATM and QRTP over Hybrid Addressing

In order to justify the use of a hybrid addressing scheme, we have to provide proof that a scheme which uses IP over ATM for both control and payload transport is not as good, in terms of performance. The hybrid addressing scheme uses IP addressing for control messages (such as setting up session), and native ATM stack for payload delivery.

7.6.1 Methodology

We compare arrival rates in the experiment. In one case the sender and receiver are connected with a raw IP socket, which is supported by IP over ATM. In the second case, on the same physical network, we have another pair of sender and receiver, programmed using Fore ATM API and linked via native ATM stack. Both cases are based on two identical ATM virtual circuits, with the same reserved bandwidth 10 Mbps. The virtual circuits pass through two Newbridge 36150 switches, and two OC-3 NIC cards. Experiments are run separately on the physical link which has no other traffic. 10000 TPDUs, each of 4000 bytes, were transmitted from sender to receiver in each case. Inter-arrival times of TPDUs are then logged and processed after the runs.

7.6.2 Analysis

The case measuring the inter-arrival time of native ATM connection reveals results matching the bandwidth reserved for the ATM virtual circuit. Figure 7.4 shows the

distribution. Average arrival interval is 0.004853 second for a 32000 bit TPDU, which translates into a 6.6 Mbps payload bandwidth. Given that a 10 Mbps ATM VC has an effective bandwidth of 9 Mbps, we have 73% average utilization. A more careful look at the distribution reveals that, by excluding some "outliners", there is approximately an average of 0.004 seconds which yields an 88% utilization.

The case measuring the interarrival time of IP over ATM connection reveals results which shows that ATM bandwidth is largely under utilized. The average arrival interval is 0.079244 second for a 32000 bit TPDU, which is a 0.404 Mbps payload bandwidth, and a 4.5% utilization of a 10 Mbps VC, which shows that it is a waste of bandwidth to allocate large bandwidth for single IP applications. A further probe into the CPU utilization of the receiver, while the sending and receiving is taking place, is only 4%. When the sending side ceases transmission, utilization and interarrival rate increase quickly.

A side-by-side comparison indicates samples of interarrival time via native ATM are much more concentrated than those via IP over ATM. This suggests sending payloads via native ATM will yield smaller jitter. This is best for multimedia applications which have strong requirements on small interarrival jitter.

To investigate the reason why IP over ATM under-utilizes the reserved bandwidth, we perform the above experiments again. In addition we turned on the ATM level statistics in each case. The observation shows that in the IP over ATM case, sender sends out average 850 ATM cells per second, which means it is sending at a 307

Kbps rate while the dedicated virtual circuit has 10 Mbps. This closely matches the measurement from the receiver which shows an average arrival rate of 0.08 second, which is 400 Kbps. We also observe that the sender sends out average 10,000 ATM cells per second, which translates into 4 Mbps, which is relatively close to the average arrival rate derived from receiver. This comparison shows that an application which sends data via IP over ATM transmits slower than those using native ATM stack. The IP layer packet rate when using native IP over ATM is about 17 times slower than when using native ATM stack, and the ATM layer cell rate when using IP over ATM is about 13 times slower.

The inefficiencies of IP over ATM may be caused by following reasons:

- Difference between software and hardware handling of communications. IP over ATM is implemented in software and is subject to processor power, and ATM stack is implemented on-board.

- Use of ATMARP table by IP over ATM.

- Fore's Use of concurrent service model in native ATM stack is superior than IP's single interrupt service routine model.

Clearly, these findings indicate the inefficiencies in IP over ATM, and they justify the use of native ATM stack for payload delivery. As well, these findings explain some earlier observations from other experiments that performance of TCP/IP-based video conferencing tool is improved by only a small amount when ATM with SONET replaces Ethernet as physical media.

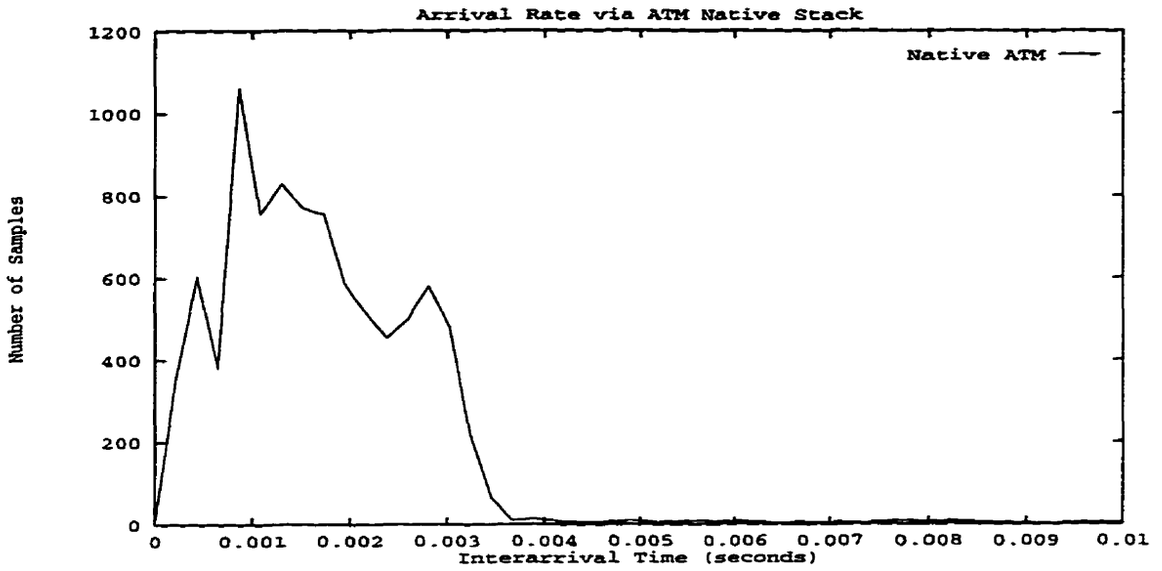

Figure 7.4: Arrival rate when using native ATM

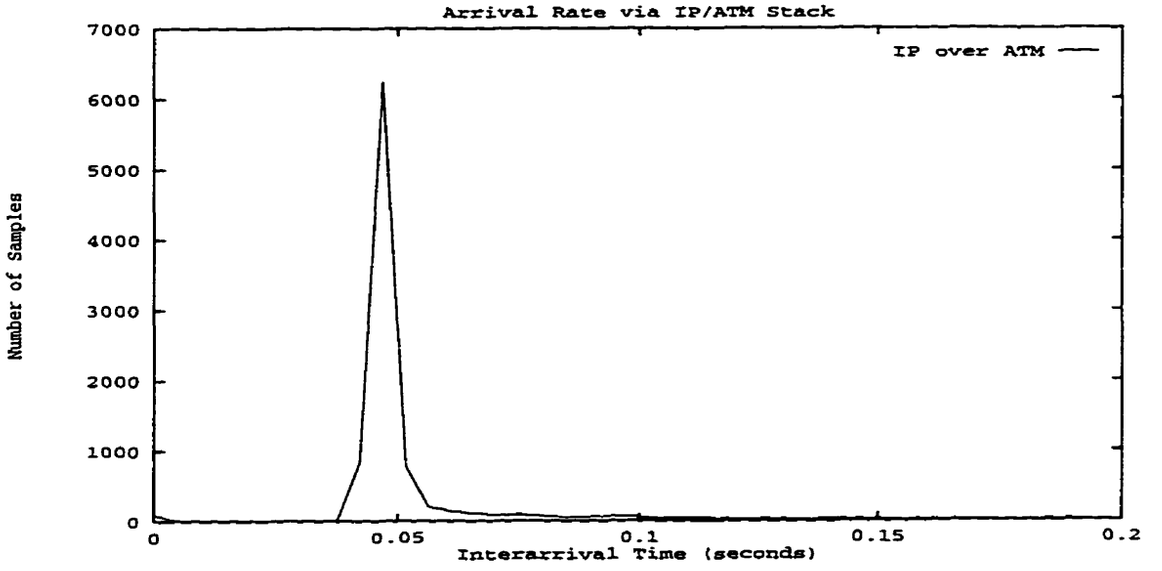

Figure 7.5: Arrival rate when using IP over ATM

7.7 Effectiveness of QoS Control

Experiments are also carried out to measure the effectiveness of QoS control. Two sessions are created with different QoS levels, and synthetic traffic are sent between two sender/receiver pairs. In the first experiment, session 1 has a higher QoS level while session 2 has a lower level. Results shows that session 1 suffers lower end-to-end delay compared to session 2. To further evaluate the mechanism, we set the QoS level of session 2 to be higher than that of of session 1. In this experiment, session 2 suffers lower end-to-end delay than session 1. Although it belongs to future work to verify how much delay is reduced with increasing of each QoS level, the qualitative conclusion that we draw from consistent results of multiple experiments is that, our control mechanism indeed ensures that the session with higher QoS suffers delay lower than those with lower QoS.

7.8 Conclusion

Our implementation performs well in terms of loss rate, QoS control and synchronization. Tuning of latency and throughput for the current prototype implementation is limited by following constraints on current implementation:

- Not being able to implement in kernel space. Current implementation is in user space and using user space memory, which is less efficient.

- Communication between application process and QRTP daemon has to be via Unix socket. If it had run in kernel space, this communication could have been changed to memory sharing controlled by interrupts.

- Using flexible size to ensure low loss even under heavy traffic load.

Chapter 8

Conclusion

8.1 Thesis Summary

The transport layer includes some of the most important functions needed for multimedia real-time communication applications, such as broadband network connection setup with QoS requirements, out-of-sequence detection and correction, and synchronization. Our experiments revealed that TCP/IP over ATM implementation does not utilize ATM bandwidth efficiently therefore other possible alternatives should be studied.

The thesis surveyed the related work on the research and development of transport middleware for high-speed networks and multimedia applications. One of the important observations is that in both high-speed networks and applications the issue of reserving resources to control or guarantee QoS is an important requirement. This means the specification and control of QoS at the transport layer should also be present to support application level QoS, and utilize the network QoS mechanisms.

In addition, a protocol development environment is needed for easy specification of a protocol stack and generation of generic code for each stack. x-Kernel is an excellent example.

The thesis introduced the design of QRTP. The data format of the protocol information exchange is dictated by the IETF RTP V2 specification. A unique structure, which consists of MCA, MA, TSAs and ASAs, has been designed and implemented. The MCA is responsible for the overall control functions, the TSAs are responsible for transporting formated data units, and the ASAs are responsible for real-time functions and user data formatting. The MA is responsible for handling management queries. A clear separation of tasks and well modularized implementation will help in future expansion of the protocol. Protocol mechanisms for addressing, multiplexing of sessions, flow control, connection management and termination, out-of-sequence detection and correction, synchronization, multicasting, and QoS control, are discussed.

The mechanisms are all implemented in a prototype, which includes a core protocol implementation, and a suite of tools and multimedia applications. The internal structure of the implementation was described in detail, and implementation issues were discussed. We drew important conclusions on issues such as whether to make a separate process at each endpoint, and whether to use one type of service agents only (TSAs) or two service agent types (TSAs and ASAs). The approach where a QRTP daemon runs on each endpoint is selected since it provides easier endpoint control, connection and resource management. We adopted ASAs on top of TSAs since it reduced the amount of work for TSAs and reduces TPDU loss.

Multimedia applications including video and audio applications were developed to test the functions of the QRTP prototype and evaluate the performance. In conclusion, QRTP provides the following main features:

- Interactive, two-way or presentation, one-way communication support;
- Quality of service support;
- Handling of multiple sessions;
- Connection-oriented support;
- Inter-session synchronization;
- Out-of-sequence detection and correction;
- A hybrid addressing and transport scheme;
- Multicasting support;
- An easy-to-use API set;
- Separation of media adaptation module and common transport module;
- Dynamic size queues for input and output;

Performance evaluation produced following findings:

- End-to-end delay is acceptable, and major components of latency have been identified to be queuing delays. The implementation modifications that can reduce latency include moving into kernel space and using fixed-size cyclic queue.

- Error rate is low.
- ATM virtual circuit bandwidth is efficiently utilized based on the measured throughput.
- Basic inter-session synchronization functions.
- Basic QoS control functions.

8.2 Contribution

This research has contributed to the field in a number of ways:

- **Evaluation of ATM native stack against TCP/IP over ATM.** We performed a evaluation on Queen's ATM testbed and came up with the conclusion that native ATM stack can yield better performance under certain circumstances. While an enormous amount of research provide performance evaluation of TCP/IP over ATM, a comparison such as ours is not available to the best of our knowledge.
- **A hybrid addressing and transport scheme.** Based on the conclusion from comparing ATM native stack and TCP/IP over ATM, we designed a hybrid addressing scheme which uses TCP/IP for control traffic and ATM native stack for payload delivery. Such a scheme avoids the overhead of IP to ATM addressing conversion and other inefficiencies.
- **Introduction of QoS levels.** QoS level specification and control was implemented in the QRTP prototype, and yields qualitative results indicating that sessions with higher QoS level always have better performance.

- **Introduction of session synchronization.** Session synchronization has been designed and implemented in QRTP prototype to provide means for synchronizing different sessions, and experiments verified that synchronized sessions suffer equal end-to-end delays.

- **Multicasting and user management functions.** QRTP implementation includes multicasting capacity and group member control functions which enables dynamic user control. Such a capacity is not currently available in normal TCP/IP implementation.

8.3 Future Work

A number of topics within this research area remain open for future work. This section discusses issues and features that deserve detailed examination.

8.3.1 Performance Tuning

A few improvements can be made to tune up the performance of the protocol. First, the implementation can be moved from the user space to kernel space. Second, the current poll-based service model can be replaced by the interrupt-based service model. Third, the flexible size, linked-list queue can be replaced by a cyclic array-based queue which will significantly reduce the queuing delays, which are main components of the overall delay.

8.3.2 Applications

More applications can be developed to use QRTP services. In return, they serve as incentives to enhance the functionality of QRTP. One such example is a remote robot control application. Such an application presents strict requirements on real-time characteristics (such as interarrival delay) on transport services. Currently video and audio sample applications are developed only to test the basic functionality of QRTP so they do not have a complete set of features such as user-friendly interfaces. These sample applications can be enhanced into more complete, user-friendly applications.

8.3.3 Transport Management

Management of QRTP is discussed in [ZGMM96]. An SNMP (Simple Network Management Protocol) MIB (Management Information Base) defined in [ZMM96-3] has been implemented in ASN.1 as an effort related to this research. A MIB compiler such as SMUT can be used to compile the ASN.1 version of the MIB into a C, which can then be used to implement SNMP agent which collects the management information from QRTP. Work is needed to convert the current non-SNMP implementation of the agent to an SNMP-compliant one.

8.3.4 QoS Levels

The current implementation of QoS level specification and control is very basic. The ultimate goal of QoS specification is to allow applications or users to specify bandwidth, delay and loss ratio. The goal of QoS control is to define and implement a set of control policies that provides guaranteed services to applications and users.

8.3.5 Compatibility

The ultimate goal of QRTP implementation is to be compatible with other implementations of RTP V2. Work is needed to experiment with the QRTP daemon and applications with other RTP V2 applications, to identify the issues in making QRTP compatible with those applications.

Bibliography

[AKS96] Ahuja, Keshav, and Saran. "Design, Implementation, and Performance Measurement of a Native-Mode ATM Transport Layer (Extended Version)". IEEE/ACM Transactions on Networking, August 1996.

[Arm95] Armitage. "Support for Multicast over UNI 3.0/3.1 based ATM Networks", Internet Draft, 1995.

[All95] Anothony Alles. "ATM Internetworking". *Cisco Technical Report*, 1995.

[And93] D.P.Anderson. "Meta-Scheduling for Distributed Continuous Media". *ACM Transaction on Computer Systems*, 11(3), August 1993.

[AT96] M. Ahmed, K. Tesink. "Definitions of Managed Objects for ATM Management Version 8.0 using SMIv2". RFC 1695, 1996.

[ATMFORUM-PNNI96] ATM Forum. "Private Network-Network Interface, Version 1.0". *ATM Forum Technical Specification*, 1996.

[ATMFORUM-UNI96] ATM Forum. "User-Network Interface Specification, Version 3.1". *ATM Forum Technical Specification*, 1996.

[ATMFORUM-TM95] ATM Forum. "Traffic Management Specification, Version 4.0". *ATM Forum Technical Specification*, 1995.

[BCB+95] M.Borden, E.Crawley, B.Davie, S.Batsell. "Integration of Real-time Services in An IP-ATM Network Architecture". RFC1821, 1995.

[BH95] R.Braden, D.Hoffman. "RSVP Application Programmig Interface (RAPI) for SunOS/BSD". Internet Draft, 1995.

[BZB93] R.Braden, L.Zhang, S.Berson, S.Herzog, S.Jamin. "Resource ReSerVation Protocol - Version 1 Functional Specification". Internet Draft, IETF, 1993.

[BFMMVZ94] Banerjea, Ferrari, Mah, Moran, Verma, Zhang. "The Tenet Real-time Protocol Suite: Design, Implementation, and Experiences". Technical Report TR-94-059, International Computer Science Institute, Berkeley, CA, November 1994.

[BFG+95] R.Bettati, D.Ferrari, A.Gupta, W.Heffner, W.Howe, M.Moran, Q.Nguyen, and R.Yavatkar. "Connection Establishment for Multi-Party Real-Time Communication". In *Proc. IEEE Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV) '95*, Durham, NH, pp. 255-266, April 1995.

[BM91] A.Banerjea and B.Mah. "The Real-Time Channel Administration Protocol". *Proceedings of 2nd International Workshop on Network and Operating System for Digital Audio and Video*, Heidelberg, Germany, November 1991.

[CBAR96] Canadian Institure for Telecommunications Research. "Canadian Broadband Applications Research Program Research Proposal", 1996.

[CBDW92] G. Coulson, G.S. Blair, N.Davies and N. Williams. "Extensions to ANSA for multimedia computing". *Computer Networks and ISDN Systems* 25 (1992) pp. 305-323.

[CDFGKSST93] D.Clark, B.Davie, D.J.Farber, I.S.Gopal, B.K.Kadaba, W.D.Sincoskie, J.M.Smith and D.L.Tennenhouse. "The AURORA gigabit testbed". *Computer Networks and ISDN Systems*, 25(1993) pp. 599-621.

[CG94] Toong Shoon Chan, Ian Gorton. "HTPNET: A High Performance Transport Protocol". To be published in *Computer Communications Journal*, Butterworth Heinemann, 1994.

[CS96] Les Y.C.Chan, J. Michel Savoie. "TCP/IP Unicast and Multicast trials for the Wide Area over ATM at BADLAB". *Proceedings of IEEE ATM 96 workshop*, Session TP, San Francisco, August 1996.

[Clark92] William J. Clark. "Multipoint Multimedia Conferencing". *IEEE Communications Magazine*, pp. 44-50. May 1992, Special Issues for Multimedia Communications.

[CT90] D.D.Clark and D.L.Tennenhouse. "Architectural considerations for a new generation of protocols". *SIGCOMM Symposium on Communication Architectures and Protocols*, pp. 200-208, IEEE, Sept. 1990. Computer Communications Review, Vol. 20(4), Sept. 1990.

[Dan92] Andre Danthine. "New transport services for high-speed networking". *Computer Networks and ISDN Systems*, 25(1992), pp. 384-399.

[Dob96] George Dobrowski. "The ATM Forum - A Point of Communication Convergence". *Proceedings of IEEE ATM 96 workshop*, Keynote Address, San Francisco, August 1996.

[DH96] Stephen E. Deering, Robert M. Hinden. "Internet Protocol, Version 6 (IPv6) Specification", RFC 1883.

[Fore96] Fore Systems Inc. "ForeThought Software Manual". Technical Manual, 1996.

[GC92] J. Green and P. Corman. "Interactive Multimedia Association Architecture Reference Model". *Interactive Multimedia Association*, Nov 1992. http://www.ctr.columbia.edu/comet/courses/ee_e9701/ee_e9701.html.

[Hui94] Joseph Hui. "Switch and Traffic Theory for Integrated Broadband Networks". *Kluwer Academic Publishers*, Fourth Printing, 1994.

[HSC+96] W.J.Hymas, H.J.Stuttgen, D.Chan, S.Sharma, S.Wise. "Socket Extensions For Native ATM Access". *Proceedings of IEEE ATM 96 International Workshop*, Session QOS, August 1996.

[IETF-AVT95] Audio-Video Transport WG. "RTP: A Transport Protocol for Real-Time Applications". Internet Draft. 1996.

[IETF-INTSERV96] Integrated Service Work Group. "Integrated Services (intserv) Charter". http://www.ietf.cnri.reston.va.us/html.charters/intserv-charter.html, January 1996.

[IETF-IPATM95] Network Working Group, IETF. "Classical IP and ARP over ATM". RFC 1577, 1995.

[Jac96] S. Jackowski. "Native ATM Support for ST2+", RFC1946, 1996.

[Jain91] R. Jain, "The Art of Computer Systems Performance Analysis". Wiley, New York, NY, 1991.

[JKV96] R. Jain, Shivkumar Kalyanaraman, and Ram Viswanathan. "The OSU Scheme for Congestion Avoidance in ATM networks Using Explicit Rate Indication". OSU-CISRC-1/96-TR02 Technical Report, Ohio State University, Dept of CIS 1996.

[Laz96] Aurel Lazar. "Tutorial: Multimedia Networking". WWW Home page http://www.ee.columbia.edu/ aurel/. January 1996.

[Laz92] Aurel Lazar. "A Real-Time Control, Management and Information Transport Architecture for Broadband Networks". *Proceedings of the 1992 International Zurich Seminar on Digital Communications*, March 16-19, 1992, pp. 281-296.

[Liao95] Willy Liao. "An Implementation of RTP (Version 1) for the x-Kernel". Technical Report. Department of Computer Science, University of Illinoise and Urbana-Champaign, August 1995.

[LLG95] Louise Lamont, Lian Li and Nicolas D. Georganas, "Centralized and Distributed Architectures for Multimedia Presentational Applications". Technical Report, Multimedia Communication Research Laboratory, Department of Electrical Engineering, University of Ottawa, 1995.

[MO95] Edwin F. Menze III, Hilarie K. Orman. "x-Kernel Programmer's Manual". Technical Manual, Department of Computer, University of Arizona, September 1995.

[MB92] Alastair J. Macartney and Gordon S. Blair. "Flexible trading in distributed multimedia systems". *Computer Networks and ISDN Systems*, 25(1992), pp. 145-157, North-Holland.

[MMTOOLS96] Multimedia Tools. http://www.rpi.edu/Internet/Guides/decemj/itools/internet-tools.html. January 1996.

[MN95] Roland Mechler and Gerald W. Neufeld, "XTP Application Programmig Interface", Technical Report, Department of Computer Science, University of British Columbia, October 1995.

[NAHR95] Klara Nahrstedt. "An Architecture For End-to-End Quality of Service Provision And Its Experimental Validations". A Ph.D. dissertation in Computer and Information Science, University of Pennsylvania, 1995.

[Ooi95] Soo Liang Ooi. "An API for Distributed Multimedia Applications". Master's Thesis, University of Waterloo, Canada, 1995.

[Pat93] Craig Patridge. "Protocols for high-speed networks: some questions and a few answers". *Computer Networks and ISDN Systems*, 25(1993)1019-1028, North-Holland.

[PLMHGM96] Perez, Liaw, Mankin, Hoffman, Grossman, Malis. "ATM Signalling Support for IP over ATM". RFC 1755, 1996.

[Ran93] P. Venkat Rangan. "Viedo Conferencing, file storage, and management in multimedia computer systems". *Computer Networks and ISDN System*, 25(1993)901-919, North-Holland.

[SMZVB92] Jurgen M. Schneider, Lothar F. Mackert, Georg Zorntlein, Roelof J. Velthuys, and Udo Bar. "An integrated enviroment for developing communication protocols". *Computer Networks and ISDN Systems*, 25(1992)305-323.

[SP96] S. Shenker, C. Partridge. "Specification of Guaranteed Quality of Service". INTERNET-DRAFT, 1996.

[Stallings93] William Stallings. *Data and Computer Communications*, Fourth Edition, McMillan, 1993.

[Sch96] H.Schulzrinne. "RTP Profile for Audio and Video Conferences with Minimal Control". RFC 1890, 1996.

[SW96] S.Sjenker and J. Wroclawski. "Network Element Services Specification Template". Internet Draft, December 1996.

[TC96] See-Mong Tan, Roy H. Campbell. "Signalling with the x-ATM Protocol Toolkit", Internal Technical Report, Department of Computer Science, University of Illinoise and Urbana-Champaign, November 1996.

[WS95] G.R.Wright, W.R.Stevens. "TCP/IP Implementation Illustrated", Volume 2. Addison-Wesley Professional Computing Series, 1995.

[VIC96] Video Conferencing Tool (vic). http://www-nrg.ee.lbl.gov/vic/, downloaded in January 1996.

[XTP95] XTP Forum. "Xpress Transport Protocol Specification", XTP Revision 6.0. Last modified August 1995.

[ZGMM96] Zhenjun Zhu, Gerald Winters, Pat Martin, H.T.Mouftah. "Management of RTP Transport Protocol". *Proceeding of 18th BIENNIAL SYMPOSIUM ON COMMUNICATIONS*, June 1996.

[ZMM96] Zhenjun Zhu, Pat Martin, H.T.Mouftah. A Real-time Multimedia Transport Protocol. 1996 TRIO/ITRC Researcher Conference, May 7-9, 1996, Queen's University, Kingston, Canada.

[ZMM96-2] Zhenjun Zhu, Pat Martin, H.T.Mouftah. "A Real-time Broadband Transport Protocol". Technical Report 1996-399, Department of Computing and Information Science, Queen's University, Kingston, Canada.

[ZMM96-3] Zhenjun Zhu, Pat Martin, H.T.Mouftah, Gerald Winters. "A Management Information Base for RTP". CD-ROM, IBM Center for Advanced Study Conference (CASCON) 1996, November 7-9, 1996, Toronto, Canada.

Appendix A

QRTP Application Programming Interface

A.1 Name

qrtpInit, qrtpJoin, qrtpSend, qrtpRecv, qrtpWithdraw, qrtpTerminate, qrtpSync, qrtpList, qrtpShutDown.

A.2 Synopsis

```
gcc [flag ...] -I$(QRTP_INC) file ... -lsocket -lnsl -lqrtp [library ...]

#include "cbartp_session.h"
#include "cbartp_access.h"

CBARTPSessionAccessInfo* qrtpInit(int req_type,
```

```
                                    char* remote_endpoint,
                                    int remote_session_id,
                                    int qos_level);
CBARTPSessionAccessInfo* qrtpJoin(int req_type,
                                    char* remote_endpoint,
                                    int remote_session_id,
                                    int qos_level);
sessionInfoList*        qrtpList(CBARTPEndPointInfo* endpoint);
int                     qrtpSync(int session1_id, int session2_id, double tolerable_skew);
int                     qrtpTerminate(int session_id);
int                     qrtpShutDown();
int                     qrtpSend(int session_id, char* buffer);
int                     qrtpRecv(int session_id, char* buffer);
```

A.3 Description

`qrtpInit` initializes a new session, whose session access information is returned by the function. The quality of service (QoS) value is indicated by `qos_level`.

`qrtpJoin` initializes a new session endpoint, whose access information is returned by the function. The quality of service value is indicated by `qos_level`. The remote endpoint, which the calling application is requesting to be connected to, is identified by a host name, `remote_endpoint`, and a session identification `remote_session_id`.

`qrtpList` returns a list of all the sessions existing on the endpoint identified by `endpoint`. It obtains this information by sending query to the MCA (Master Control Agent) at the host identified.

`qrtpSync` synchronizes two sessions whose session identifications are `session1_id` and `session2_id`, respectively. A tolerable skew, `tolerable_skew`, is defined in seconds.

`qrtpTerminate` terminates a session with session identification as `session_id`.

`qrtpShutDown` terminates the process MCA.

`qrtpSend` sends data `buffer` via session which has idenficication `session_id`. `qrtpRecv` receives data `buffer` via session which has idenficication `session_id`.

A.4 Return Values and Errors

For `qrtpInit` or `qrtpJoin`, if return value is NULL, then the initialization process fails. For `qrtpList`, if the return value is NULL, then the number of sessions is 0. For Other functions, if the return value is 0, then the operations are successful.

Appendix B

Protocol Data Unit Formats

- **Version (V)**: 2 bit. RTP specification regulates that this field represent the RTP versions.

- **Padding (P)**: Indicates whether there are padding in the payload that should be ignored by playback application. If there is, the *last* octet of the payload will indicate how many octets, counting from the bottom, are paddings. This is significant since real-time

  applications require great responsiveness, and even if the payload size is fixed at an optimal level, there are still occasions when payload is not filled when it has to be sent. Other times lower layer PDU is larger and RTP has to use paddings to fill a fixed block lower level PDU to have the payload carried.

- **Extension (X)**: Indicates whether *Header Extension* is to be used to carry internal implementation-specific information that is ignored by other RTP implementations. We set this bit to 0 for now. The header extension can be used for our future header information extension.

- **CSRC Count (CC)**: Number of CSRC identifiers for a PDU.
- **Marker (M)**: Used to indicate events like video or audio frame boundaries.
- **Payload Type (PT)**: A mapping from payload type to encoding and decoding. For current research, we will choose PT=14 (MPA Audio) and PT=26 (JPEG Video), and temporarily PT=96 for medical data.
- **Sequence number**: Sequence numbers are stamped onto RTP PDUs while they are sent out and will be used to check lost PDUs and restore sequence at the receiver end. This is one of the most important part of a real-time transport protocol.
- **timestamp**:Used for two synchronizations: synchronization between sender and receiver(s) without receivers back pressuring sender, and synchronization between data flows on different sessions.
- **SSRC**: This is the identification of the source. It is *not* the normal network address but rather a long randomly generated number to ensure that *no* two sources have the same SSRC identifier within each session.
- **CSRC**: Optional. This lists contributing sources such as mixers and translators introduced in RTP specifications. This is very useful when there are different links with different bandwidths in a wide area network and it is not useful in a broadband network. Currently unused and CC set to 0.
- **Payload**: Encoding and decoding of payload is either defined by standard body, if registered PT types are used, or defined by individual implementations.

Appendix C

Session Data Structure and Protocol Control Block

```
typedef struct CBARTPSessionStruct
{
        int                     sessionId;
        int                     mtu;
        int                     maximumPayloadSize;
        CBARTPEndUser           *owner;
        CBARTPEndUser           *userList;
        CBARTPSessionAgent      *send_agent;
        CBARTPSessionAgent      *recv_agent;
        CBARTPSessionRemoteRecvAgentList
                                *remote_agent_list;
        int                     numberOfUsers;
        rtpPduQueueType         *TpduQueue;
```

```
rtpPduQueueType         *RpduQueue;
/*
 * When synchronization is done, synchronized sessions
 * share the same queue, which is a merged syncMergedQueue.
 * R-TSAs will enqueue incoming TPDUs into this queue and
 * R-ASAs will dequeue from this queue.
 */
rtpPduQueueType         *syncMergedQueue;
void*                   synclist;
struct CBARTPSessionStruct
                        *next;
/*
 ******************************
 * Protocol Control Block - PCB
 ******************************
 */
/*
 * Information updated by the MCA
 */
unsigned long           sessionStartUpTime;
int                     recvWin;
/*
 ************************************************************
 * Information ONLY updated by CBARTPSessionPerfEnterTASA,
 * collected when a PDU is generated by sender ASA agent.
 ************************************************************
```

```
 */
/*
 ************************************************************
 * Information ONLY updated by CBARTPSessionPerfEnterTRTTRANS,
 ************************************************************
 */
int                     totalSentPdus;
double                  currentWindowTotalSendQueueDelay;
/*
 ************************************************************
 * Information ONLY updated by CBARTPSessionPerfEnterRRTTRANS,
 ************************************************************
 */
int                     currentWinExpectedPdus;
int                     currentWindowActPdus;
int                     totalExpectedPdus;     /* Received */
int                     totalActPdus;          /* Received */
unsigned long           totalPDULost;
/*
 ************************************************************
 * Information ONLY updated by CBARTPSessionPerfEnterRASA,
 * collected when a PDU is generated by sender ASA agent.
 ************************************************************
 */
double                  currentWindowTotalRecvQueueDelay;
double                  lastPduTime;
```

```
        unsigned long           currentInterPDUInterval;
        unsigned long           currentInterPDUIntervalJitter;
        unsigned long           lastRecvPDUSeq;
        unsigned long           totalRecvErrorPDUs;
        unsigned long           totalRecvOutOfSeq;
        /*
         ************************************************************
         * Others
         ************************************************************
         */
        int             numBurst;
        int             totalBurstDur;
        unsigned long   sessionUpTime;
        float           lastRecvTPDUSentTime;
        int             QoSLevel;
        int             atm_send_fd;
        int             atm_recv_fd;
        int             atm_vpi;
        int             atm_vci;
        int             use_atm_flag;
} CBARTPSession;
```