

Winter 1998

# A Framework for Controlling Quality of Sessions in Multimedia Systems

Alaa S. Youssef  
*Old Dominion University*

Follow this and additional works at: [https://digitalcommons.odu.edu/computerscience\\_etds](https://digitalcommons.odu.edu/computerscience_etds)



Part of the [Computer Sciences Commons](#)

---

## Recommended Citation

Youssef, Alaa S. "A Framework for Controlling Quality of Sessions in Multimedia Systems" (1998). Doctor of Philosophy (PhD), dissertation, Computer Science, Old Dominion University, DOI: 10.25777/8dwt-5z34  
[https://digitalcommons.odu.edu/computerscience\\_etds/83](https://digitalcommons.odu.edu/computerscience_etds/83)

This Dissertation is brought to you for free and open access by the Computer Science at ODU Digital Commons. It has been accepted for inclusion in Computer Science Theses & Dissertations by an authorized administrator of ODU Digital Commons. For more information, please contact [digitalcommons@odu.edu](mailto:digitalcommons@odu.edu).

# A FRAMEWORK FOR CONTROLLING QUALITY OF SESSIONS IN MULTIMEDIA SYSTEMS

by

Alaa S. Youssef

B.Sc. June 1991, Alexandria University, Egypt

M.Sc. April 1994, Alexandria University, Egypt

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


DOCTOR OF PHILOSOPHY


COMPUTER SCIENCE

OLD DOMINION UNIVERSITY

December 1998


Approved by: . . .

  
Hussein Abdel-Wahab (Director)

  
Kurt Malv (Director)

  
Shunichi Toida (Member)

  
Christian Wild (Member)

  
Mohamed Gouda (Member)

# ABSTRACT

## A FRAMEWORK FOR CONTROLLING QUALITY OF SESSIONS IN MULTIMEDIA SYSTEMS

Alaa S. Youssef  
Old Dominion University, 1998  
Co-Directors: Dr. Hussein Abdel-Wahab  
Dr. Kurt Maly

Collaborative multimedia systems demand overall session quality control beyond the level of quality of service (QoS) pertaining to individual connections in isolation of others. At every instant in time, the quality of the session depends on the actual QoS offered by the system to each of the application streams, as well as on the relative priorities of these streams according to the application semantics. We introduce a framework for achieving QoSess control and address the architectural issues involved in designing a QoSess control layer that realizes the proposed framework. In addition, we detail our contributions for two main components of the QoSess control layer. The first component is a scalable and robust feedback protocol, which allows for determining the worst case state among a group of receivers of a stream. This mechanism is used for controlling the transmission rates of multimedia sources in both cases of layered and single-rate multicast streams. The second component is a set of inter-stream adaptation algorithms that dynamically control the bandwidth shares of the streams belonging to a session. Additionally, in order to ensure stability and responsiveness in the inter-stream adaptation process, several measures are taken, including devising a domain rate control protocol. The performance of the proposed mechanisms is analyzed and their advantages are demonstrated by simulation and experimental results.

Copyright ©1998, by Alaa S. Youssef, All Rights Reserved.



## ACKNOWLEDGMENTS

There are many people who have contributed to the successful completion of this dissertation. My deepest gratitude and appreciation are due to Dr. Hussein Abdel-Wahab, for his persistent direction and constructive guidance throughout this work. I would like to express my sincere thanks to Dr. Kurt Maly, for his great help and valuable directions. I extend my thanks to all of my committee members for their patience and guidance on my research, especially Dr. Mohamed Gouda from the Computer Sciences Department at The University of Texas at Austin. Also, I would like to thank all the faculty and colleagues at the Computer Science Department of Old Dominion University for their support and encouragement.

I am truly blessed to have a wonderful supportive family. My understanding and encouraging wife, Samah, my loving son, Ahmed, and my beautiful daughter, Salma, were strong motivations for me to work hard to fulfill this goal. Although miles have often separated me from my parents and sister, they have been a constant source of love and encouragement. My deepest thanks to my parents, who have inspired me throughout my life, and to my sister, Aliaa. I am very thankful to my entire family for their unfailing belief in my ability.

# TABLE OF CONTENTS

	Page
LIST OF TABLES	viii
LIST OF FIGURES	ix
Chapter	
I INTRODUCTION	1
1.1 Overview . . . . .	1
1.2 Environment and Applications . . . . .	3
1.2.1 Interactive Remote Instruction: an example application . . . .	5
1.3 Objectives . . . . .	8
1.4 Framework . . . . .	12
1.5 Outline . . . . .	14
II BACKGROUND AND RELATED WORK	16
2.1 Quality of Service . . . . .	16
2.1.1 Proactive solutions . . . . .	18
2.1.2 Reactive solutions . . . . .	24
2.2 Scalable Feedback Techniques . . . . .	34
III INTER-STREAM BANDWIDTH ADAPTATION	39
3.1 Elements of Inter-Stream Bandwidth Adaptation . . . . .	40
3.2 The Quality of Session Graph . . . . .	44

3.2.1	Mapping graphs to priority classes . . . . .	44
3.3	Inter-Stream Adaptation in a Reservation Environment . . . . .	47
3.3.1	Resource allocation models . . . . .	49
3.3.2	RISA: An optimizing approach for range-based resource al- location . . . . .	51
3.3.3	I-WFS: A fair approach for range-based resource allocation .	54
3.3.4	A metric for comparing resource allocation policies . . . . .	55
3.3.5	Preliminary simulation results . . . . .	56
3.4	Traffic Characterization . . . . .	61
3.5	Bounding Delays . . . . .	63
3.5.1	Bounding delays in a FCFS scheduler . . . . .	63
3.5.2	RISA and I-WFS revisited . . . . .	66
3.5.3	Simulation results . . . . .	67
3.6	Inter-Stream Adaptation in a Best-Effort Environment . . . . .	71
3.6.1	Results and evaluation . . . . .	73
3.7	Conclusions . . . . .	78
IV	STATE FEEDBACK PROTOCOL . . . . .	80
4.1	Feedback Role in Adaptive Multimedia Multicast Systems . . . . .	81
4.2	A Scalable Feedback Mechanism . . . . .	84
4.2.1	Selecting the timeout functions . . . . .	85
4.2.2	Exploring the parameter space . . . . .	87
4.2.3	Estimating the round-trip time . . . . .	90
4.3	Simulation Study and Performance Comparison . . . . .	90
4.3.1	Bounding constants in timing function . . . . .	91
4.3.2	Evaluating the round-trip time estimation technique . . . . .	95
4.3.3	Performance comparison . . . . .	95
4.4	Enhancing the Feedback Mechanism . . . . .	101
4.4.1	Adaptive feedback . . . . .	101

4.4.2	Passive feedback . . . . .	104
4.5	Conclusions . . . . .	105
V	ARCHITECTURAL DESIGN, PROTOTYPE AND EXPERIMENTAL RESULTS	106
5.1	Design Principles and Architecture Overview . . . . .	107
5.2	Design Details . . . . .	110
5.2.1	Monitoring agents . . . . .	111
5.2.2	Inter-stream adaptation agents . . . . .	119
5.3	Stability Provisions . . . . .	123
5.3.1	Inter-stream adaptation state machine . . . . .	123
5.3.2	Multi-modal timers . . . . .	125
5.3.3	Learning network delay . . . . .	128
5.3.4	Domain rate control protocol . . . . .	129
5.4	Evaluation . . . . .	138
5.5	Experimental Results . . . . .	141
5.6	Conclusions . . . . .	152
VI	CONCLUSION AND FUTURE EXTENSIONS	155
6.1	Conclusion . . . . .	155
6.2	Future Extensions . . . . .	159
	REFERENCES	163
	APPENDICES	
	A. ANALYSIS OF THE A-IWFS ALGORITHM . . . . .	170
	B. MONITORING AGENT CLIENT INTERFACE . . . . .	173
	C. ACRONYMS . . . . .	178
	VITA	179

## LIST OF TABLES

TABLE	Page
3.1 Notation used in the A-IWFS algorithm . . . . .	73
5.1 Monitor/ISA interface messages . . . . .	116
5.2 ISA/SM interface messages . . . . .	121
5.3 Heuristics for determining the mode of an ISA agent . . . . .	128
5.4 Comparison of receiver driven rate control protocols . . . . .	140

## LIST OF FIGURES

FIGURE	Page
1.1 QoSess control in an MBone session. . . . .	4
1.2 QoSess control in a gateway architecture. . . . .	5
1.3 Connections going through the teacher's node in IRI. . . . .	6
1.4 QoSess control layer and flow of control information. . . . .	12
2.1 Overhead of session messages. . . . .	37
3.1 Elements of inter-stream adaptation. . . . .	40
3.2 Representing the relative priorities of streams. (a) QoSess graph. (b) Priority list. (c) Priority classes. . . . .	43
3.3 The G2P mapping algorithm. . . . .	46
3.4 Two-step mapping of a QoSess graph to priority classes. (a) Compute node depths. (b) Assign priorities. . . . .	47
3.5 QoSess CDF variation with load for RISA. . . . .	57
3.6 QoSess CDF variation with load for average fixed-point. . . . .	58
3.7 QoSess in an IRI derived scenario. . . . .	58
3.8 Effect of resource availability on QoSess for RISA. . . . .	59
3.9 Resource utilization in an IRI derived scenario. . . . .	59
3.10 Bounding functions of three traffic characterization models. . . . .	62
3.11 Effect of peak-to-average ratio on acceptance ratio. . . . .	67
3.12 Effect of burst size on acceptance ratio. . . . .	68
3.13 Variation of maximum delay with acceptance ratio. . . . .	68

3.14	Effect of delay bound on acceptance ratio. . . . .	69
3.15	Effect of delay bound on utilization. . . . .	69
3.16	Effect of delay bound on QoSess. . . . .	70
3.17	The A-IWFS algorithm for resource allocation. . . . .	74
3.18	Comparing bandwidth allocation to I-WFS. (a) A-IWFS. (b) Amir's algorithm. . . . .	75
3.19	Comparing bandwidth allocation by Amir's algorithm to RISA. . . .	76
4.1	Distribution of timeout periods according to receiver state. . . . .	86
4.2	The extreme topologies. (a) Star. (b) Chain. . . . .	88
4.3	The effect of $C_2$ on the number of replies. (a) Chain topology. (b) Star topology. . . . .	92
4.4	The effect of $C_2$ on response time. . . . .	93
4.5	Accuracy of RTT estimate. (a) Chain topology. (b) Star topology. . .	94
4.6	Performance of the chain topology for $t=0$ . . . . .	96
4.7	Performance of the chain topology for $t=0.2$ . . . . .	97
4.8	Performance of the star topology for $t=0$ . . . . .	98
4.9	Performance of the star topology for $t=0.2$ . . . . .	99
4.10	Effect of adaptive feedback. (a) Number of replies. (b) Response time. . . . .	102
5.1	Architecture of the Quality of Session control layer. . . . .	109
5.2	Components of the monitoring agent. . . . .	111
5.3	The Stream and Monitor data structures. . . . .	112
5.4	Components of the inter-stream adaptation agent. . . . .	120
5.5	Inter-stream adaptation protocol state machine. . . . .	124
5.6	Constant timers and stability. . . . .	142
5.7	Smoothed losses and stability. . . . .	143
5.8	Multi-modal timers and stability. . . . .	143

5.9	Aggregate throughput at a receiver on a congested LAN. (a) Without QoSess. (b) With QoSess. . . . .	146
5.10	Losses at a receiver on a congested LAN. (a) Without QoSess. (b) With QoSess. . . . .	147
5.11	Jitter at a receiver on a congested LAN. (a) Without QoSess. (b) With QoSess. . . . .	148
5.12	Aggregate throughput at a receiver behind a bottleneck link. . . . .	149
5.13	Adapting the sleep timeout ( $T_s$ ) to large buffers. (a) Aggregate throughput. (b) Sleep timeout ( $T_s$ ). . . . .	151
5.14	Adapting the sleep timeout ( $T_s$ ) to large delays. (a) Aggregate throughput. (b) Sleep timeout ( $T_s$ ). . . . .	153



# CHAPTER I

## INTRODUCTION

The capabilities of the Internet have increased at a phenomenal rate since its inception. However, the user base and application requirements have increased at a much greater rate. It is clear that simply increasing network bandwidth is not sufficient for handling the growth in Internet resource demands. Instead, network and application designers must develop and utilize new mechanisms that foster efficient use of the available resources. In this dissertation, we present our view and efforts in order to achieve this goal.

### 1.1 Overview

Explosive growth in the deployment of new network technology has stimulated tremendous interest in *Interactive Multimedia Collaborative (IMC)* applications. IMC applications are being developed for distance learning and training, scientific and engineering cooperative efforts, tele-meetings, and Internet games, to mention just a few areas. However, the real time requirements of the continuous media streams, deployed by IMC applications, demand special treatment. A key issue that characterizes most of the approaches taken for handling the requirements of continuous media streams, is the management of the *Quality of Service (QoS)* offered to individual connections in isolation of others. Our approach, however, is to dynami-

---

The journal model for this dissertation is the *IEEE Transactions*.

cally control the QoS offered by the system across the set of connections belonging to the IMC application, in order to avoid any potential competition for resources among the streams that comprise a session. This control is based on the application semantics, with the objective of maintaining the best overall quality of session, at every instant in time, despite potential fluctuations in resource availability. We use the term *Quality of Session (QoSess)* to denote this QoS support across multiple connections.

Consider for example, a two way audio-video conferencing application. Such an application may choose to degrade the quality of video only, while maintaining the audio fidelity, in reaction to network congestion. A system which does not accommodate this inter-stream relationship, may degrade the performance of all streams with an equal proportion, in an attempt to react to the overload situation in a fair way.

Moreover, an application may have instantaneous priorities for its streams that vary over time. Building on the same example mentioned above, if the conversation was going on between two physicians, and at a certain point in the teleconference, the video image of one of the participants was replaced by a VCR tape playback of an operation, then the application may prefer a degradation in the quality of the audio rather than that of the video, in reaction to any overload situations.

In order to scale to large groups of participants, modern IMC applications rely on IP multicast [23]. In such a heterogeneous environment, the need to compromise the quality of one stream in favor of another may not only be due to temporary congestion situations, but it may be due to system inherent capacity constraints. For example, a video conferencing application that supports several simultaneous participants, may not find enough network bandwidth, or system processing capabilities, to provide a full-motion video stream of each participant. Instead, a participant may receive a full-motion video image of the current speaker, besides low frame rate video images of the rest of the participants.

The previous examples backup our proposition that IMC systems demand

overall quality control across connections, beyond the level of QoS pertaining to individual connections in a given session. The instantaneous quality of the session not only depends on the offered QoS for each of the streams, but on the relative importance of these streams from the application perspective as well. An end-to-end monitoring mechanism can best capture the actual QoS offered to individual connections, as perceived by the end-user. Additionally, in order to react to static bottlenecks or dynamic congestion in the network or end-hosts, inter-stream adaptation mechanisms are necessary to regulate the consumption of resources among the streams in a given session. Thus, monitoring and inter-stream adaptation are the two main building blocks of the QoSess control framework. This framework is not only useful for best-effort networks, but for networks with resource reservation capabilities as well. In the latter case, QoSess control is needed to manage the allocation of resources which are collectively reserved for the streams of a distributed application.

## 1.2 Environment and Applications

The *Multicast Backbone (MBone)* constructs a virtual multicast topology on top of the existing Internet [29]. Considering the size of the Internet, wide-scale upgrade of every router and host to provide multicast support was impossible. Thus, the MBone was constructed by incrementally upgrading hosts and routers to support multicast. These multicast-enabled nodes were interconnected with point-to-point IP-encapsulated tunnels. A tunnel consists of one or more non-multicast hops and is viewed by multicast routing protocols as a single logical link, which is statically configured. Currently, most of the multicast routing is performed by protocols, which implement source-based shortest-path trees, such as the *Distance vector multicast routing protocol (DVMRP)* [24]. However, this approach to multicast routing suffers from scaling problems, and hence, the *Center-based routing* approach was recently introduced [25]. In Center-based protocols, a single entity in the network

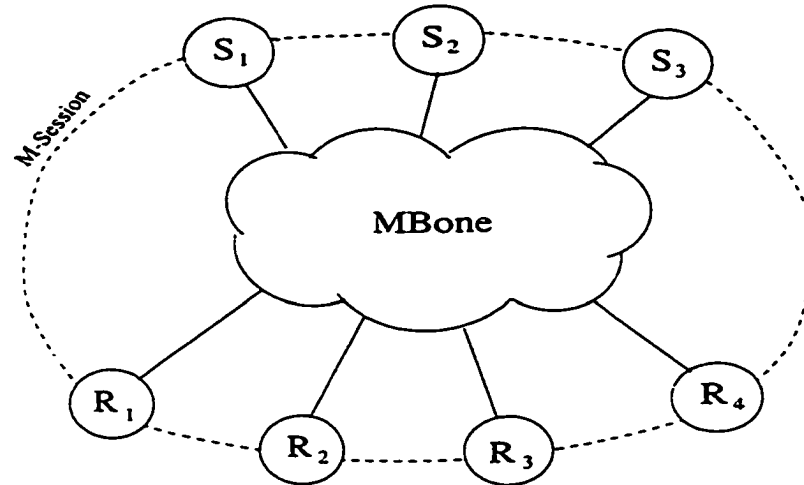


Fig. 1.1. QoS control in an MBone session.

is designated to receive messages from all sources for delivery to a set of receivers.

From an application perspective, the MBone enables the application entities to establish multicast communication channels, independent of their geographical locations. As shown in Figure 1.1, the MBone hides the multicast network topology and routing details from the application sending and receiving entities. One of our main objectives is to engage these application entities, which compose a session (denoted by M-Session in the figure), in end-to-end protocols and mechanisms that hide heterogeneity in network and end-host capabilities from the application.

A common configuration for multimedia conferences entails one or more users participating across a bottleneck link with the rest of the users participating across a higher performance network region. Media gateways are one approach for managing network bandwidth in these environments [3]. As shown in Figure 1.2, in this setup, a media gateway (typically operating at the application level) manages the bottleneck link by rate limiting the media streams, possibly through transcoding. Gateways are not only used for crossing bottleneck links, but for crossing security and addressing boundaries as well. Firewalls and home-user network connections which do not support multicast addressing are not uncommon in the Internet, and

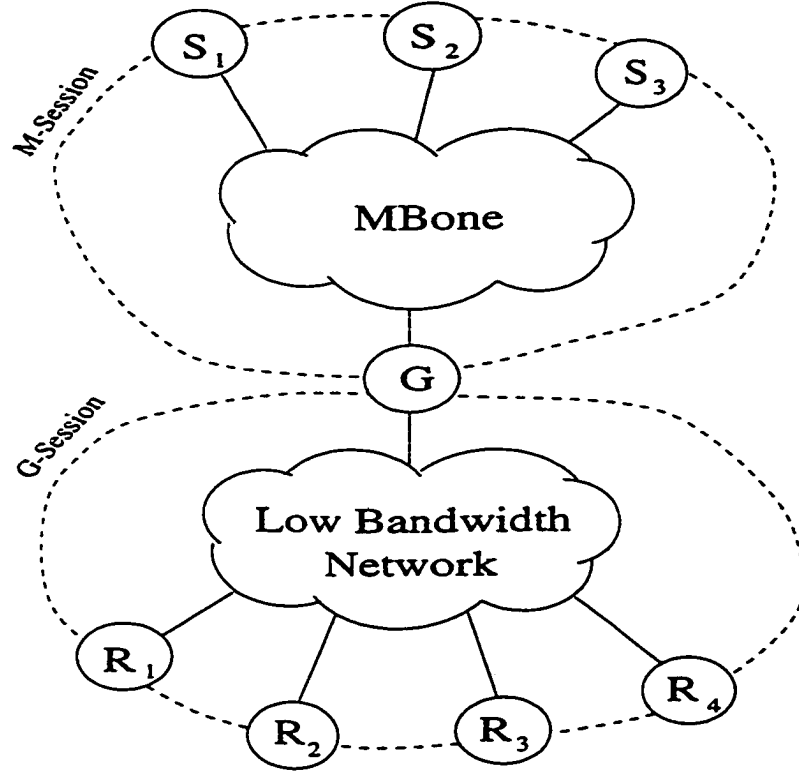


Fig. 1.2. QoS control in a gateway architecture.

gateways provide the means for crossing such boundaries. In this setup, two instances of the QoS mechanisms are needed in order to dynamically control potential congestion conditions: one instance controls the M-Session as before, where the gateway participates as a receiver; and another instance controls the G-Session in which the gateway participates as the source of all streams.

In the following subsection, we describe an example IMC application which would benefit from QoS control in the illustrated operating environments.

### 1.2.1 Interactive Remote Instruction: an example application

An example IMC application that can directly benefit from the work presented in this dissertation is the *Interactive Remote Instruction (IRI)* system [2, 45]. IRI

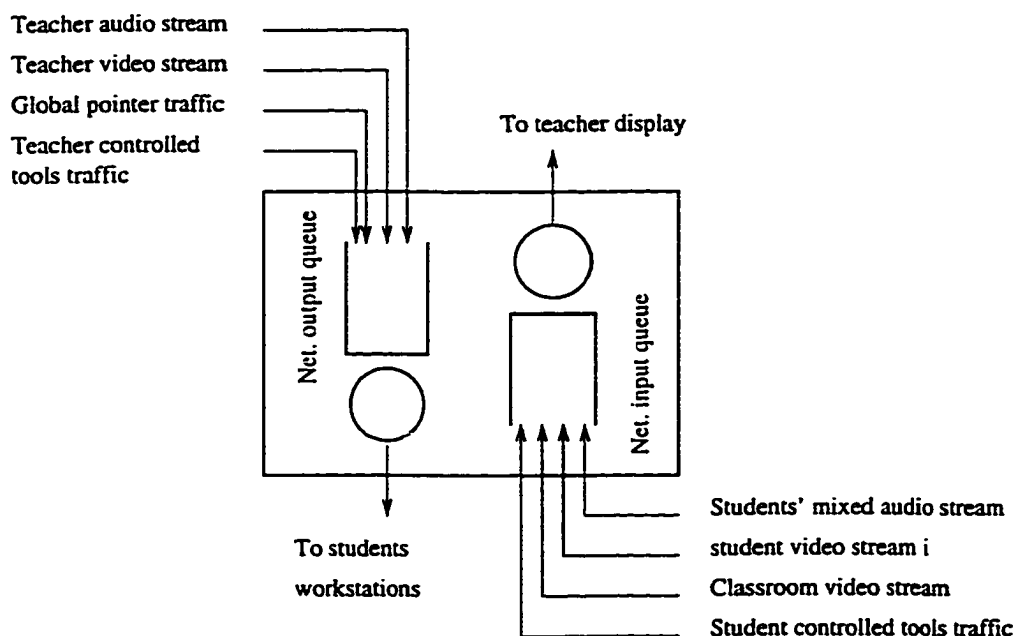


Fig. 1.3. Connections going through the teacher's node in IRI.

is an integrated learning environment that supports distance learning in a virtual classroom. It provides the users with a simple interface and encourages both teacher/student and student/student interaction through two-way audio and video, and tool sharing. IRI employs a multi-point to multi-point communication pattern that is based on multicasting, and it does not tolerate excessive degradation in QoS.

IRI employs a group of continuous and discrete media streams that cooperate to provide the overall integrated view to the participants of a session, who use currently available best-effort networks. The typical operating environment for IRI is a set of distributed classrooms, each consisting of a number of workstations interconnected by a 10 Mbps LAN (typically Ethernet). This set of local networks is interconnected through a backbone that reserves 10 Mbps of bandwidth for IRI. All members of the session (teacher and students) are intended to have an identical view of all streams. This synchronization of views elevates the distance boundary, and helps all students to share the same experience regardless of their physical location.

In Figure 1.3, we present a teacher's workstation with the emphasis on the

communication activity from that node. The out-going connections from the host machine are teacher video and audio streams, collaborative tool generated data, and global pointer movements. The incoming connections into the host include student video streams, classroom video streams, incoming audio streams, and data generated from student controlled tools. The following are typical QoS requirements of each of these connections. These requirements are not too stringent, and a range of operation, around the points listed below, is acceptable.

*Teacher video.* Requires 640 *pixel* X 480 *pixel*, 15 frames per second video, which is delay-sensitive and can afford moderate loss of data.

*Teacher audio.* Generates 64 Kbps data, and is highly sensitive to delay and losses.

*Student video streams.* Require 320 *pixel* X 240 *pixel*, 10 frames per second video which is delay-sensitive and can afford moderate loss of data.

*Student audio streams.* 64 Kbps connections which are highly sensitive to delay and losses.

*Classroom video streams.* Require 320 *pixel* X 240 *pixel*, 5 frames per second video which is delay-sensitive and can afford heavy losses.

*Global pointer movements.* When the teacher moves the pointer on the screen, the same should be visible on the other participants' screens. This application is also delay-sensitive, but has no stringent requirements on losses. Even if we loose some of these packets, the application will not suffer.

*Data-transfer applications or student controlled tools.* These are typical data transfer applications which mainly require throughput. They are not too sensitive to delays and cannot afford to loose data.

IRI is a typical IMC application which motivates the need for adaptive performance over best-effort networks and operating systems. IRI demands group management of the resources across multiple connections in the application.

A QoS guarantee is a global issue and will be defeated if any of the intermediate networking components fails to guarantee it. This is likely to happen not only in best-effort networks, but in networks providing enhanced best-effort services, such as differential services [28], and in networks providing statistical or predicted guarantees. Regardless of the used network service model, intelligent adaptation is the way to constantly maintain the best possible session fidelity.

### 1.3 Objectives

We have recognized several issues that arise in distributed collaborative multimedia applications, that employ groups of streams, which cooperate to provide an integrated view to the users. Some of these issues were addressed separately, while others were barely addressed by researchers. Our objective is to provide a unified approach for addressing this set of issues, which are summarized below.

#### **Supporting heterogeneity in network connections and host capabilities**

The group of streams needed by the application at a certain instant in time may require more resources than the capacity of some of the network connections, or the capability of some of the hosts. Examples of such resources are network bandwidth, buffers, or CPU power to serve the generation/display of the group of streams. Sacrificing some of the streams will solve the problem in this case, at the expense of losing the information contained in those streams. An alternative solution would be to operate some of the streams at a degraded performance level (e.g., lower video frame rate). While such degradation decisions can be made statically for a certain environment, with a pre-specified network and hosts setup, yet the portability of the distributed application to other setups is jeopardized, unless it is built on top of an



intelligent layer that provides services to the application such as making adaptation decisions in a seamless manner. This problem is compounded by the simultaneous co-existence of hosts and network connections with heterogeneous capacities in the same session. This implies that any possible solution to the problem must take a receiver-oriented approach in order to accommodate such a heterogeneous environment.

### **Avoiding congestion and host overload conditions**

Even if the resource requirements of the application streams do not exceed the system resources, yet at a certain instant in time a congestion in the network or an overload in the CPU of an end host may take place.

Congestion in the network can originate at a leaf subnet due to high loads, that result in longer access times for the shared medium. It can also appear as longer queuing delays and eventually packet drops at intermediate network nodes, due to the high loads offered at these routing points. In both cases, congestion manifests itself in the form of packet delays and losses. These in turn affect the quality of the stream as perceived by the end user.

Overload in the end system can originate due to several reasons. High I/O interrupt rates and running many simultaneous processes are common reasons. Regardless of the reason, overload in the end host leads some of the processes running on the host to suffer from lack of CPU time to perform the required processing. Overload in the end host manifests itself in the form of packet delays and losses due to excessive queuing delays in the end host.

Whether an overload in the end host or a congestion in the network occurs, the end application suffers from packet delays and eventually packet losses. Hence the symptoms are the same although the causes may differ. The IMC application must be prepared to deal with these situations in real-time. As mentioned earlier, although several researchers have attempted to address the problem of adapting the performance of a stream in the face of overload conditions, we believe that the group

of streams belonging to an IMC session must be managed together from a global view.

### **Accounting for the cooperative nature of streams**

One of the most important motivations for this work is realizing the fact that the group of streams which belong to an IMC session are intended to cooperate in order to provide the end user with a complete integrated view. If each stream is managed in isolation of the others, especially while attempting to avoid or react to congestion conditions, we may end up with a set of competing streams; each trying to get as much resources as possible at the expense of the needs of other streams. This, in turn, may cause an overall low perceived quality of session. On the other hand, if a global view exists and the group of streams is managed in a consistent manner with a primary goal of providing the best attainable session quality at all times, better user perception of the session can be achieved. This aspect is related to the following issue.

### **Accommodating application dynamic behavior**

The group of streams belonging to an IMC application have a highly dynamic nature. A stream may be started or stopped at any instant in time. Additionally, the relative priority of a stream with respect to the other streams varies with time. The priority of a stream is a measure of the contribution of that stream to the total view presented to the users. The instantaneous priority of a stream is a function of the set of active streams at that instant. This implies that an adaptation decision taken at instant  $t$ , where stream  $s_1$  has higher priority than stream  $s_2$ , should favor  $s_1$  at the expense of degrading the performance of  $s_2$ , rather than degrading the performance of both streams by equal amounts. The actual amount by which the operation level of one or more of the streams is degraded should be a function of the effective QoS offered to the streams.

## Utilizing group reservations efficiently

Recently, several approaches for providing group reservations with guarantees were proposed [35, 48, 67]. This mode of reservation gives flexibility to the IMC application to perform group management of the allocated resources by assigning them to different streams at different instants throughout the session lifetime, in a way that efficiently utilizes the reserved shared resources. It is desirable to have such resource management tasks performed automatically by a service layer whose operation is guided by minimal specifications from the application.

Although several researchers have addressed the problem of adapting multimedia streams to react to capacity constraints or overload situations [12, 41, 55], yet these efforts were directed towards managing single streams in isolation of others. Also, none of those efforts attempted to benefit from the cooperative nature of the streams belonging to an IMC application, not to mention accommodating the dynamic behavior of those streams.

The purpose of this work is to provide a service layer that is accessible by the IMC application. It is called the *Quality of Session (QoSess) control layer*. The objective of this layer is to enforce the best attainable instantaneous quality of session. It supports the cooperative nature and dynamic behavior of the streams of an IMC application, accounts for heterogeneity in capacity constraints, and adapts the streams in order to react to congestion situations. The success of this layer in performing its allotted tasks will help distributed collaborative multimedia systems that are being increasingly deployed in the fields of education, training, engineering design, co-authoring, etc., to achieve their goals and operate successfully with the best attainable overall session fidelity, over networks that provide merely best-effort services. As more advanced network service models become available, the role of the QoSess layer shifts more towards supporting application dynamics and efficient utilization of resources shared among the streams of an IMC session.

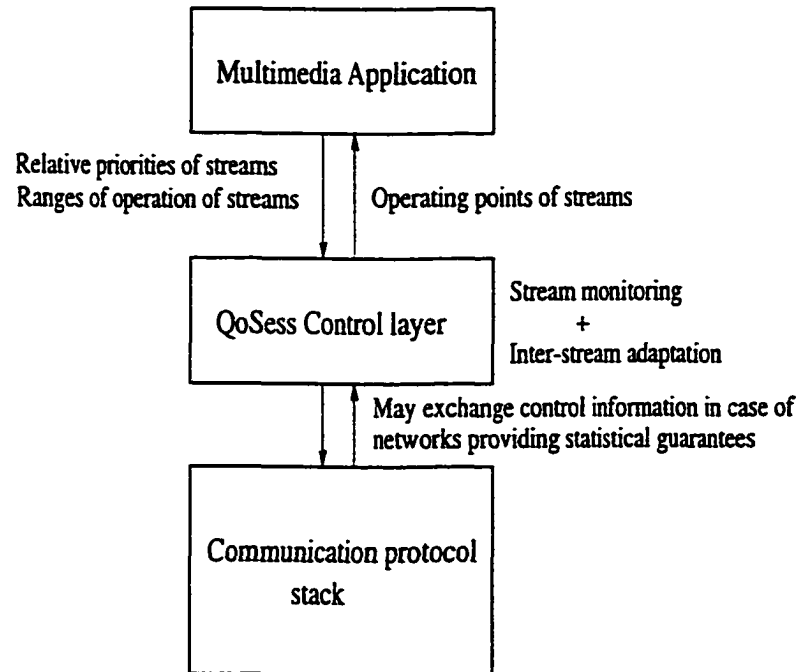


Fig. 1.4. QoSess control layer and flow of control information.

## 1.4 Framework

In the previous section, we discussed the main issues that commonly arise in distributed IMC applications, and we illustrated an example system, IRI, which would benefit from a unified solution that addresses these issues. In this dissertation, we propose a framework for collectively addressing these issues. The framework is represented by a *Quality of Session (QoSess) Control Layer* [65], which is introduced between the distributed multimedia application and the communication protocol stack, as shown in Figure 1.4. This layer need not be a monolithic unit embedded in the communication stack, and may be realized using more than one architecture. In Chapter 5, we present one such architecture [62].

The role of the QoSess layer is to enforce the best attainable session quality at every instant in time throughout the lifetime of the session. It adapts the streams to avoid congestion situations, accounts for capacity constraints, and respects the semantic requirements of the application.

The proposed framework for achieving QoS control relies on the ability of the IMC application to adapt, i.e., the ability to dynamically change the rates of the different streams. In addition, in order to support heterogeneity of receivers and network connections, multi-grade streams are centric to the framework. Multi-grade transmission can be achieved either by hierarchical encoding [47, 52], or by simulcast which is the parallel transmission of several streams each carrying the same information encoded at a different grade [42, 59].

In IMC applications, unreliable uncontrolled transmission is typically used for video, audio, images, and some data streams (e.g., pointer movement data). The QoS control layer acts as a closed loop feedback system that constantly monitors the observed behavior of the streams, makes inter-stream adaptation decisions, and sets the new operating level for each stream from within its permissible range of operating points. Over a wide area network, in the presence of a resource reservation protocol such as RSVP [67], the QoS control layer manages the resources that are collectively reserved for the streams of a distributed application. In order to achieve its goals, the QoS control layer has to perform the following tasks.

**End-to-end monitoring.** The actual QoS offered to each stream is estimated from the view point of every receiver of that stream. This requires a monitoring agent to be associated with every receiver as well as with the sender. Views of the QoS provided to a certain stream may vary from one receiver to another. This is due to the fact that the group of receivers may have network connections or hosts with different loads and capacities. The monitoring mechanism used must be of minimal overhead, must not introduce an extra level of data copying, and should rely as much as possible on feed-forward information in order to scale. If feedback is necessary, then a carefully chosen scalable feedback mechanism should be used. In Chapter 4, we present a new scalable and robust feedback protocol [63].

**Inter-stream adaptation.** Information obtained from the monitor reports, to-

gether with knowledge about the priorities of the streams with respect to each other, is used by an inter-stream adaptation (ISA) unit to select the level of operation of each stream. These operation levels are selected with the objective of optimizing the overall quality of the session. In Chapter 3, we present and analyze several inter-stream bandwidth adaptation algorithms [61, 66].

**Interfacing to the application.** In order to perform its tasks, the QoSSess control layer must exchange several pieces of information with the application. These are listed below.

- The application provides the QoSSess layer with the permissible operating range of each stream.
- The application provides the QoSSess layer with relative priorities of the active streams. This information is provided asynchronously at any change in priorities of streams based on application semantics.
- The QoSSess layer provides the application with the operating level of each stream. The ISA decisions are made as often as necessary, and the relevant application entities are notified whenever there is a change in the operating point of a stream.

The QoSSess control layer plays an end-to-end role and extends the functionality of the transport layer. In order to expedite its deployment, it must be easy to embed within the application, in a manner that conforms to the concept of application level framing [18].

## 1.5 Outline

We present our work as follows. Chapter 2 describes work related to the problem under consideration. In Chapter 3, we focus on inter-stream adaptation techniques, and present algorithms for sharing the available bandwidth among the streams that

cooperate to compose a session. In Chapter 4, we present a protocol for providing state feedback information to the source of a multicast stream in a scalable and robust manner. This protocol is used in adjusting the transmission rate of the source, in order to prevent any waste in resource usage. A proposed architecture for realizing the quality of session control framework is presented in Chapter 5, together with the protocols necessary for achieving stability and responsiveness in the control process. Finally, in Chapter 6, we summarize the contributions and future extensions of our work.

## CHAPTER II

### BACKGROUND AND RELATED WORK

There are two major approaches for addressing the special requirements of multimedia streams. First, the proactive approach, where a resource reservation protocol and underlying scheduling mechanisms work to reserve and guarantee end-to-end resources. Second, the reactive approach, where application entities (senders and receivers) adapt to the level of available resources. In Section 2.1, we outline the basic techniques proposed in the literature for realizing each of these two approaches, after briefly introducing the concept of *Quality of Service (QoS)*.

A common problem that often arises in multicast systems in general, and in multimedia multicast systems specifically, is the need for soliciting feedback information from a group of receivers, in a scalable manner. The Quality of Session framework proposed in this dissertation is no exception, and indeed this problem arises in it and needs to be addressed. A brief survey of the different approaches for providing scalable feedback is presented in Section 2.2.

#### 2.1 Quality of Service

The concept of *quality of service (QoS)* has been introduced, in order to characterize what is acceptable for a communication client [14, 17, 39]. The QoS is defined by a set of service parameters (e.g., throughput, delay) requested by the client, and by the degree of commitment made by the system to maintain these parameters.



Systems can be broadly classified into the following main classes, with respect to the level of QoS guarantees that they provide [14, 17, 39].

**Deterministic.** In this case the system is committed to give hard guarantees for the bounds on all the service parameters. Hard real-time applications require such guarantees from the system.

**Predicted or statistical.** In this case the parameters are statistically guaranteed by the system (e.g., the delay is guaranteed to be less than the requested value with probability 90%). Such guarantees are suitable for soft real-time and multimedia applications. Although some guarantees may be violated, yet the multimedia application need not do a lot of adaptation, as the service is provided at an acceptable level most of the time.

**Best-effort.** In this case, no guarantees are provided. While typical data transfer applications, such as *ftp*, *e-mail*, *telnet*, *etc.*, can run well on such systems, yet for multimedia applications to run on such systems there is a great need for intelligent adaptation, to be exercised at the application level, in order to provide the users with streams at acceptable quality.

**Enhanced best-effort.** These are traditional best-effort systems whose service model is augmented by basic mechanisms which favor real-time traffic when degradation in performance is inevitable due to increased loads. For example, in the *Differential Services* [28] approach, which is currently under investigation by the IETF (Internet Engineering Task Force) for deployment in the Internet, real-time packets are tagged. These tags enable intermediate routers to identify these packets and to process them before others, thus allowing these packets to meet their deadlines without explicit reservations. Also, these tags allow a router suffering from congestion to drop packets other than those tagged as real-time as much as possible before having to drop any of the real-time packets.

There is a trade-off between the level of QoS guarantees provided by the system and the application complexity. At the top level, with deterministic QoS guarantees, enough resources are allocated so that the application can simply play-back the original signal, without the need for any adaptation. At a lower level, with statistical QoS guarantees, the receiver will be able to play-back the original signal with acceptable quality most of the time, assuming that enough resources are allocated. It is up to the application to decide whether to do adaptation to face the eventual degradations in the level of service, or to totally ignore the problem. At the bottom level, in best-effort systems, an application would need to support different levels of quality for the play-back signal, and to dynamically adapt to match the changes in the level of offered service.

Several proactive solutions were proposed, in the literature, in order to provide guaranteed QoS support for multimedia streams. Also, several reactive solutions were presented in order to accommodate the requirements of multimedia streams over best-effort networks. The following two subsections explore the major solutions in each category.

### **2.1.1 Proactive solutions**

Several attempts to define proactive QoS architectures were reported in the literature. We give a brief overview of the major proposals in what follows.

#### **Quality of Service Architecture (QoS-A)**

In order to support continuous media in multi-service networks, a layered architecture of services and mechanisms for QoS management (QoS-A) was proposed by Campbell et al. [14, 20, 21, 39]. QoS-A integrates a range of QoS configurable protocols and mechanisms in both the network and the end-system. At the network level, these include network reservation protocols, and service disciplines capable to implement the required QoS. At the end-system, QoS-A relies on various mechanisms,

such as thread management and buffer allocation. These mechanisms are implemented by extending the existing abstraction of the Chorus micro-kernel operating system [37], to include QoS configurability, connection oriented communications, and real-time threads [20, 21]. QoS-A is structured in three planes: *protocol*, *QoS maintenance*, and *flow management*. The *QoS maintenance* plane contains a number of QoS managers, that are responsible for guaranteeing the appropriate QoS level for each flow. The *flow management* plane includes flow admission control, resource reservation and QoS-based routing.

For the purpose of resource reservation the network is partitioned into domains, called *flow management domains*. A flow management domain consists of an arbitrary collection of network devices, such as routers, multimedia workstations, and continuous media storage servers. In each domain there is a *resource server* that is responsible for the domain flow management, and resource allocation. When the resource server receives a reservation request, it consults its local representation of the domain resource availability. If the request can be satisfied, then the flow management plane provisionally marks the required resources as allocated, and then sends a multicast request to all QoS managers of all nodes in the path(s) from source to destination. Upon receiving this message, each QoS manager tries to allocate the requested resources. If it succeeds, then it sends a confirmation message to the resource server. When the resource server receives the confirmation messages from all QoS managers, the reservation request is granted.

While QoS-A provides a framework to specify and implement the appropriate QoS requested by multimedia streams, it deals with those streams as competing entities that must be isolated from one another. It does not provide any means for the group management of resources of related streams. Also, it does not provide support for advance reservations or graceful service degradation.

## Resource Reservation Protocol (RSVP)

The resource reservation protocol, RSVP [48, 67], is a receiver-initiated control protocol that reserves resources for *simplex* data streams, in an integrated services packet switching network, such as the Internet. Every server consists of an *RSVP daemon* that executes the reservation protocol, a *packet classifier* that classifies incoming packets according to the QoS class they belong to, and a *packet scheduler* that guarantees the required resources in order to meet the QoS parameters. An RSVP reservation request consists of a pair: *flowspecs* and *filterspecs*. The *flowspecs* specifies the parameters for the desired QoS, while the *filterspecs* defines the set of data packets from the input stream to receive the QoS specified in *flowspecs*. In this way, it is possible to select arbitrary subsets of packets in a given session; generally, such subsets might be defined in terms of any fields in any protocol headers in the packet. Any packets that belong to one session but do not match any of the *filterspecs* are sent as best-effort traffic.

The reservation algorithm proceeds as follows. The receiver initiates a reservation by sending a request message containing the *flowspecs* and *filterspecs* upstream to the sender. Upon receiving this message, an RSVP daemon determines whether the request could be granted. If this test fails, then a rejection message is sent back to the initiator. If the request is granted, then the RSVP daemon passes the *filterspecs* to the packet classifier, and the *flowspecs* to the packet scheduler. Next, the request is propagated up-stream to the next hop. Upon receiving a rejection message, the RSVP daemon informs the packet scheduler, and the packet classifier, and as a result all the resources are freed. Further, the rejection message is propagated to the initiator.

The main advantage of the RSVP protocol is that it can be easily integrated with the existing Internet protocol suite. In addition, RSVP supports group reservations. A reserved path can be used by several senders, either simultaneously through the use of *wild-card filterspecs*, or one at a time, without the need for initi-

ating a new reservation, through the use of *dynamic filterspecs*. These two modes of reservation give flexibility to the application to perform group management of the allocated resources by assigning them to different streams at different instants in time throughout the lifetime of a session. However, RSVP does not address this group management problem and leaves its responsibility to the application level. Also, an inherent problem with RSVP is the big amount of soft state information that must be maintained and refreshed at every router along the path of a connection. This state information is accessed with each incoming packet leading to a degradation in router throughput. The problem is profounded for backbone routers through which thousands of connections exist simultaneously.

### Tenet protocol suite

The Tenet protocol suite provides guaranteed communication services [6, 30, 69]. The protocol suite consists of five protocols: three data delivery protocols, and two control protocols. The data delivery protocols are the real-time Internet protocol (RTIP), the real-time message transport protocol (RMTP), and the continuous media transport protocol (CMTP). RTIP is the network layer protocol, while RMTP and CMTP are transport layer protocols that provide message oriented and stream oriented transport services, respectively, on top of RTIP. The two control protocols are the real-time channel administration protocol (RCAP), and the real-time control message protocol (RTCMP). RCAP is responsible for establishment, tear-down, and modification of the channels. RTCMP is responsible for control and management during data transfer. This protocol suite is the first set of communication protocols that can transfer real-time streams with guaranteed quality in packet-switching inter-networks. In this scheme, channels are set up in an establishment phase that precedes the data transfer phase. An establishment message is issued from the source of the channel and travels to the destination, causing admission tests to be done in each node along the path, and resources to be tentatively reserved by RCAP. Then the reverse pass of the establishment begins. An accept message is sent hop-by-

hop back to the source. At each node on the path, the local RCAP may choose to relax the reservations of resources that were over-reserved on the forward pass. When invoking RCAP, the client must specify its performance requirements (QoS parameters), and the worst case traffic parameters.

In Tenet Suite-II, major enhancements were added, the most important and relevant of which are: the use of a multicast channel as the basic channel abstraction; introducing ranges for the traffic and performance bounds instead of forcing the parameters to take on single values only; allowing for advance reservations; and adding a sharing group abstraction [7]. In most multi-party communication scenarios only a small subset of the potential senders are active at a given time. The sharing group abstraction allows clients to describe such behavior to the network allowing for sharing of resource allocations between such related channels. Admission control tests and packet scheduling mechanisms use a *group traffic specification*, which indicates the maximum combined traffic entering the network from all channels of the group rather than the traffic specifications for the individual channels. This gives the distributed multimedia application flexibility in managing the reserved resources and allocating them to the appropriate senders at every instant in time throughout the session lifetime. Like RSVP, the Tenet protocol suite leaves to the application the responsibility of managing the allocation of the reserved group traffic among different sources.

### **Quality of service architecture for native TCP/IP over ATM (QUANTA)**

In QUANTA [27], a *ripple-through* classification mechanism was proposed, whereby applications were classified depending on their QoS parameters, unlike other approaches where these QoS parameters are grouped into different categories. In addition, the concepts of generic soft state (GSS) and current GSS (CGSS) were presented to accommodate the dynamics of the application, reduce the QoS mapping losses across different QoS architectures, provide group and individual identities of a connection, and aid in feeding the current status of the application communi-

cation path. In QUANTA, the feedback from the network is intended to be used to dynamically alter the QoS requirements of the application. The success of the approach is measured by verifying that the application stays within the requested range of operation (*ROP*), despite load variations.

Although a main component of QUANTA's approach is the usage of monitoring and feedback information to adapt the resource requirements of a stream in order to maintain the QoS provided within the application's requested ROP, yet it assumes a static contractual ROP that is specified and respected for each stream independent of all other streams that are owned by the application. Hence it does not account for the dynamic change of priorities of streams along the execution time of the application. Also, this feedback and control is done at the network level, and thus may not reflect the application's end-to-end point of view regarding the perceived QoS. Additionally, the applicability of the mechanism in multicast systems was not addressed.

### Dynamic QoS Management (DQM)

Dynamic QoS Management (DQM) was introduced for the control of hierarchically encoded flows, in heterogeneous multimedia networking environments [13]. An adaptive network service was proposed for the transmission of multi-layer coded flows. The service offers hard guarantees to the base layer, and fairness guarantees to the enhancement layers. The guarantees provided to the enhancement layers are based on allocating bandwidth according to a weighted fair sharing (WFS) policy. Weights are statically derived from the maximum bandwidth requirements of the streams. In DQM, *QoS filters* manipulate hierarchically coded flows as they progress through the communication system. *QoS adaptors* scale flows at the end systems based on the flow's measured performance, the available bandwidth, and user supplied scaling policy. *QoS groups* provide baseline QoS for multicast communications. This approach was taken as a compromise between the inaccurate statistical modeling of variable bit-rate (VBR) video, and the inefficiency in utilizing bandwidth when

using constant bit-rate (CBR) video models.

After providing hard guarantees to the base layers of all streams, the residual bandwidth is divided according to the WFS policy among the enhancement layers of all streams. The available bandwidth for a stream is provided to the adaptors by the network service. In addition the adaptor at the source receives feedback from the receiver's end transport entity about the delay, jitter, and losses. The adaptors control filters that are located at the source, destination, and in intermediate network nodes based on this information.

Although the concept of adaptation was introduced in DQM, yet it is a completely network oriented approach that regards admitted streams as competing entities. The application states only the scaling policy for the stream at initialization, and leaves the adaptors and filters to manage the encoded output based on the available bandwidth dictated by the network service. This bandwidth allocation is based on the pure static nature of the streams and does not capture any of the dynamic changes of priorities of the streams relative to each other according to the application state. Thus, DQM carries the ROP concept one step further, by committing to adhere to allocating the bandwidth available for enhancement layers in proportion to the bandwidth requirements of the admitted streams.

### 2.1.2 Reactive solutions

In what follows, we briefly summarize the main reactive schemes proposed for handling the real-time requirements of multimedia streams.

#### **Bell labs multimedia experiments over Ethernet**

At Bell labs, the workstation and network performance, during the exchange of multimedia streams over Ethernets, was investigated [26]. The effects of end-system protocol overhead, end-system CPU load, and Ethernet load on the end-to-end delay and breaks in the multimedia streams, were analyzed. The study concludes that,



when the protocol processing time in the workstation is large (e.g., 3.5 msec), the delays in the workstation dominate the Ethernet LAN delays, which appear as limited noise, in the end-to-end delay, that can be easily controlled. This noise is less effective with the migration to faster networks. The other source of variability in the end-to-end delay is the workstation processing and is also easily controlled by minimal packet buffering. On the other hand, when the protocol processing time is small (e.g., 1 msec), the protocol overhead delays are comparable to the Ethernet delays. In this case, the computation of the amount of buffering needed to smooth the play-out should consider the LAN delays as well. Two mechanisms were proposed: aggregation of the streams at the application layer for transport; and use of differential play-out offsets for the streams. Differential play-out ameliorates the differences in encoding and decoding times for video and audio. The aggregation of the streams simplifies the task of play-out synchronization to a great extent. Aggregation is also intended for decreasing the total number of packets sent per second, and hence decreasing the protocol packet processing overhead. This is particularly important in the case of using a low bit rate video encoding scheme, like H.261 [15], which provides a constant bit-rate (CBR) video stream of as low as 64 kbps, resulting in small frame sizes (264 bytes every 33 msec). With most of the higher bit rate encoding methods, the message sizes are bigger than the Ethernet maximum packet transmission unit. In these cases, aggregation may not yield significant reduction in packet rates, while introducing a level of complexity to the application, as it becomes responsible for demultiplexing the streams. Also, aggregation is useful only when the streams originate from the same source, which may not be always true.

There are two basic results to conclude from this work. First, the feasibility of real-time transportation of multimedia streams over best-effort networks and end-systems. Second, although both the network and end-system contribute to the end-to-end delay and jitter, yet the main contributor may vary according to the amount and type of load on the network, and the capabilities of the end-system. This suggests that, in order to judge the quality of the multimedia streams in hetero-

geneous environments, there is a need for an end-to-end application-level monitoring facility that is capable of observing the overall quality of the streams, as perceived by the end users, after accounting for all network and end-systems potential overheads.

### **Multimedia transport protocol (MTP)**

In [41, 55], an application-level transmission control framework was introduced for continuous media transmission over best-effort networks. The framework is built on the concept of selecting an appropriate operating point from a set of feasible operating points for a particular stream. The operating points are described as pairs of bit rate and message rate. The message rate (which is inversely proportional to the message size) at the entry point to the transport layer is considered as a proportional estimate to the packet rate generated by the network. The study concentrated, in part, on the effect of varying the message size for a media stream. The authors identify a number of constraints that help in bounding the space of feasible operating points [55]. The constraints can be divided mainly into perceptual constraints and network constraints. Perceptual constraints include minimum required bandwidth and end-to-end delay, in order to achieve acceptable perception of the stream. Network constraints are further divided into static structural capacity constraints and dynamic congestion constraints. An adaptive scheme which is capable of dynamically selecting a feasible operating point, within the limits of the above constraints, was devised.

Although adaptation was the main vehicle used in this work to provide acceptable QoS to the application over best-effort networks, yet the authors focused on intra-stream adaptation. Inter-stream adaptation was not considered.

The effect of increasing the level of fragmentation is known to have a big impact on the end hosts as well as on the intermediate routers. In IP, intermediate routers are not capable of performing re-assembly, and hence all hosts sitting behind the link with the minimal MTU (maximum transmission unit) will receive packets of at most that size. Considering a standard Ethernet with MTU=1500 bytes,

the value of the MTU is small enough leaving no room for thinking about using messages of smaller size, especially for video applications. Moreover, the utilization of the network is also an important factor that should be always maintained at a high level. One way of trying to minimize losses in bandwidth, as well as in processing power at end and intermediate hosts, is to generate messages of size equal to the minimum MTU along the path from source to destination. This implies avoiding any fragmentation/re-assembly overhead.

We argue that it is better to fix the message size for each stream, rather than using it as one of the control factors. Counter example situations can be found where decreasing the message rate (increasing the message size,  $m$ ) may lead to an increase in the actual packet rate sent over the network. If  $m \geq p$ , where  $p$  is the MTU of the network, then increasing  $m$  by any non-integer multiple of  $p$  implies an increase in the packet rate, rather than the intended decrease. However, if correct integral multiples of  $p$  are used for increasing  $m$ , while the bit rate of the stream is kept constant, then we are actually increasing the burstiness of the stream. On the other hand, if  $m < p$ , and decreasing the message rate (increasing the message size) is allowable within the perceptual constraints imposed, then the choice of  $m$  was not optimal from the efficiency point of view. A bigger value for  $m$  is more appropriate for such stream to avoid any unnecessary load on nodes along the path of the stream, as well as to avoid any waste in bandwidth resulting from packet headers overhead. In both cases, there is no point in increasing the message rate (while keeping the bit rate constant) as long as the perceptual constraints of the stream do not impose such a high rate.

In summary, we see that the maximum message size should be determined by the perceptual constraints and the smallest MTU along the path of the stream packets. Once the size is determined, it should be fixed and not used as a control parameter. Indeed, in IPv6, the disadvantage of sending messages of sizes bigger than the MTU were realized, and an IP source is no longer allowed to send messages bigger than the MTU of the whole path from source to destination [38].

## **Real-time transport protocol (RTP)**

In [12], the real-time transport protocol (RTP) and the real-time control protocol (RTCP) associated with it, were used to control the transmission rate of a video source, in response to network congestion. RTP is a thin protocol providing support for applications with real-time properties, including timing reconstruction, loss detection, security and content identification [50, 51]. These services are not provided by existing end-to-end transport protocols, and hence RTP is introduced to fill this gap complementing existing transport protocols. While UDP/IP is its initial target networking environment, efforts have been made to make RTP network-independent. RTP is also currently in experimental use directly over ATM adaptation layers. RTP does not address the issue of resource reservation; instead, it relies on resource reservation protocols such as RSVP, if available.

RTCP messages are multicast periodically to the session participants. RTCP messages are used for QoS monitoring and congestion control. Since they are multicast, all session members can survey how the other participants are performing. Sources, that have recently sent audio or video packets, send periodic sender reports. These contain timing information useful for inter-media synchronization as well as cumulative packet and byte counts that allow receivers to compute the sender rate. All receivers send periodic receiver reports, corresponding to each of the sources they heard from recently. These reports contain the highest sequence number received, the fraction of packets lost, packets inter-arrival jitter, and timestamps. For conferencing applications, RTCP messages contain an SDES (source description) packet, containing detailed information about the participants: a canonical name; user name; email address; telephone number; application specific information; and alert messages. RTCP messages are periodic and generated by all participants. Hence a mechanism is applied for scaling the control traffic load with data traffic load so that it makes up a certain percentage (5%) of the data rate.

In the experiments mentioned above, a feedback control mechanism was de-

vised based on RTCP control reports that are periodically multicast from each receiver. The RTCP control reports include information that enable the calculation of packet losses. On receiving an RTCP receiver report, the source performed network state estimation, and possibly a consequent bandwidth adjustment decision was made.

A scalability problem is inherent to this sender-based adaptation approach. The sender computes losses for every receiver from information in the RTCP receiver reports. Increasing the number of receivers beyond certain limits may overload the sender with the amount of computation it needs to do. In addition, the solution adopted to limit the bandwidth occupied by RTCP reports to a fixed percentage of the data rate (5 %) implies slower reaction to congestion as the group of participants grows larger.

The bandwidth adjustment algorithm based its decisions on the reported bandwidth,  $B_r$ , and not on the allowed bandwidth,  $B_a$ . This caused a low rate video scene showing a black screen to decrease the value of  $B_r$  and hence  $B_a$ , and then it took the system 100 seconds to restore the bandwidth again to higher levels suitable for normal scenes. Thus, the maximum of both  $B_a$  and  $B_r$  is the factor which should be used in any adaptation process.

Experiments conducted with two or more sources showed the state of competition among these sources. This state became very obvious when each source was started at a different rate, in which case each of them adapted to maintain this rate, and the stream started at a lower rate could not acquire bandwidth higher or even close to the one started at a higher rate. This competition between the streams, in addition to accounting only for the network state in the adaptation algorithm, while neglecting the potential dynamic change of priority of each stream from the application point of view, are important issues that need to be accounted for in a more elaborate and scalable approach to the problem of application-level control over the amount of bandwidth available to the streams owned by the application.

### Destination set grouping (DSG)

In an attempt to avoid the scalability problems of sender-based congestion control and adaptation, destination set grouping (DSG) was proposed [16]. DSG is based on the concept of *simulcast*, which is the parallel transmission of several streams each carrying the same information encoded at a different grade [42, 59]. DSG partitions the receivers into groups where each group receives one of the parallel streams at a rate suitable to all members of the group. A receiver starts in the slowest group, and progressively moves to faster groups until it reaches the group with the fastest rate it can handle. If the receiver has knowledge about the bandwidth available to it, it can start immediately in a high group. In addition to the inter-group flow control, where a receiver moves from group to group until it tunes to a suitable rate, intra-group bandwidth control is deployed by DSG. Through a feedback mechanism, receivers within a group may influence the source, within specified limits, to change that group's rate to match the slowest receiver in the group. If a receiver cannot get satisfactory rate within a group, it moves to another group.

Unfortunately, the simulcast approach of DSG sacrifices some of the scale gain that initially prompted the use of multicast, because duplicate data is sent for each group. In [42], several enhancements for the basic DSG protocol were proposed in order to limit the overall bandwidth wasted by this redundancy in the multicast streams. However, the splitting of the transmitted data into layers, as opposed to splitting the receivers into groups, overcomes this drawback and hence is more appealing. Protocols for layered-data multicast are discussed below.

### Receiver-driven layered multicast (RLM)

The use of layered encoding schemes enables multicast-based communication protocols to deliver optimal quality to receivers with heterogeneous capabilities. In layer encoding schemes a stream is separated into a *base* layer and one or more *enhancement* layers. The base layer can be independently decoded, while the enhancement

layers can only be decoded in presence of the base and lower enhancement layers information. While many researchers have recently proposed different layered media encoding schemes (e.g., [47, 52]), MPEG-2 is the only standard that supports layered encoding by defining four scaling modes: spatial, temporal, SNR, and data partitioning [36]. Layering in MPEG-2 can be achieved using one or more of these modes.

RLM [46] extends the best-effort IP multicast model to achieve receiver adaptability to capacity and congestion constraints. The source takes no active role in the protocol. It merely transmits each layer to a separate multicast address. Conceptually, each receiver runs a simple control loop: on congestion, drop a layer; on spare capacity, add a layer. In order to determine whether spare capacity exists or not, each receiver periodically probes for higher bandwidth by joining the next layer up. This is called a *join experiment*. If congestion is detected immediately after joining a layer, the join experiment is considered to be a failure and a join timer associated with this layer is backed-off. Join experiments are coordinated among all receivers in the group by explicit announcements. This allows for *shared learning* from the outcome of a failed join experiment. Those receivers which did not perform the join experiment by themselves, but detected congestion after hearing about the start of an experiment correlate that congestion to the experiment and back-off their relevant join timers, accordingly.

The fully distributed approach to rate control taken by RLM, where a receiver performing a join experiment makes an announcement to the whole group, has several drawbacks. First, the load introduced by making announcements to far receivers that could not possibly benefit from the experiment is unjustified. Second, these announcements from far receivers may confuse others if they correlate overloads coincidentally developing in their domains to the active join experiment in a different domain. Finally, even within the same domain, confusion may happen because the receiver whose join experiment failed does not explicitly announce that. Thus, other receivers may correlate an overload to the active join experiment, while

the fact that the joiner did not suffer from that overload clearly implies that this overload is due to other conditions developing at the host which detected it. The LVMR protocol described below attempts to solve some of these problems.

### **Layered video multicast with retransmission (LVMR)**

In the LVMR protocol [43, 44], rate control by shared learning among receivers of a multicast stream is achieved through installing a set of managers arranged in a hierarchy that matches the structure of the multicast routing tree rooted at the sender. In this way, correct correlations between join experiments and congestion resulting from these experiments can be made across several subnets. However, while the hierarchical structure fits naturally a sender and a group of receivers for one stream, it does not fit the nature of a distributed session in which multiple senders co-exist. Another drawback of this approach is the need for knowing the structure of the multicast routing tree, in order to install the intermediate managers appropriately. Such an interaction between LVMR and routing protocols is not defined. At the least, knowledge about the full network topology is necessary for the proper arrangement of the managers.

In LVMR, join experiments are synchronized; a receiver may only join any specific layer at certain times determined using a simple modulus function of the last frame number received from the base layer. This ensures that join experiments for the same layer are totally overlapping. Although this is necessary when probing for bandwidth beyond the current highest rate received in a certain subtree, it imposes unnecessary limitations on low-end receivers (or loaded hosts) attempting to join lower layers.

The LVMR protocol, however, has an advantage over the other proactive and reactive approaches for handling continuous media streams; it allows for retransmission of lost packets, provided that the estimated retransmission time is within an allowed delay bound. Although it is universally accepted that delay bounds in live real-time sessions do not allow for significant gains from retransmission, this fea-



ture is particularly useful for playback applications where the delay bound can be relaxed by buffering at the session startup time, or even at a pause in the middle of the session.

### **Sender-initiated congestion control for layered multicast**

In [57], a congestion control mechanism for layered multicast data streams is presented. As in the other proposals for layered multicast, the mechanism relies on router suppression of undesired layers in order to alleviate congestion in over-loaded subtrees. However this mechanism differs from the others in two regards. First, it uses explicit synchronization points, which are specially marked packets in the data stream, to synchronize receivers actions of joining any of the layers. Second, instead of receiver-initiated probing for availability of bandwidth, a sender-initiated probing technique is used. These sender-initiated probes consist of the periodic generation of short bursts of packets, followed by an equally long relaxation period during which no packets are sent. The bursts have the effect of a join attempt. For the duration of the burst, the bandwidth used is effectively doubled; if the bottleneck bandwidth is not sufficient, queues build up and eventually losses occur. Such congestion signals are not interpreted as a signal to lower the subscription level, but rather as hints for not increasing the subscription level.

The only claimed advantage of these sender-initiated probes over actual receiver join experiments is that the burst duration can be controlled. However, since this control is performed on the sender side, and since the network delay may vary from one receiver to another, this control is hard to balance with potentially widely varying receivers views. Short bursts may not be sufficient to induce the desired congestion signals as they may be absorbed by buffering in the network, while long bursts may last more than receiver-initiated probes for some of the receivers. Besides the undesired jitter deliberately introduced into the media streams by the sender, the way these bursts are introduced leads to doubling the sender rate for the burst duration. For high rate streams, this load may cause serious congestion. Obviously,

receiver-initiated probing by adding only a single layer at a time would not produce such high fluctuations in performance. Finally, as with LVMR, join experiments for receivers receiving rates below the highest cumulative rate entering the domain need not be synchronized. In this approach, these joins are not only synchronized with similar joins, but possibly with higher layer joins as well, which may lead to ambiguous results for the low rate join experiments. This is due to miss-interpretation of congestion developing because of the parallel higher layer joins. On the other hand, the reported advantage for this protocol is that it shares bandwidth fairly with TCP connections along the same network path. This makes this protocol more suitable for rate control in reliable data multicast [56]. The high burstiness and delay variations introduced by the sender-initiated probes makes it less suitable for continuous media streams.

## 2.2 Scalable Feedback Techniques

Soliciting information from receivers in a multicast group might create a *reply implosion* problem, in which a potentially large number of receivers send almost simultaneous feedback messages that contain redundant information. In [9], a survey of the classical approaches to address this problem was presented. These approaches are summarized below.

**Probabilistic reply.** In a probabilistic reply scheme, a receiver responds to a probe from the source with a certain probability. If the source does not receive a reply within a certain timeout period, it sends another probe. This scheme is easy to implement. However, the source is not guaranteed to receive the worst news from the group within a certain limited period. In addition, the relationship between the reply probability and the group size is not well defined.

**Expanding scope search.** In the expanding scope search scheme, the time-to-live (TTL) of the probe packets sent by the source is gradually increased. This

scheme aims at pacing the replies according to the source capacity of handling them, since the source does not re-send the probe with increased scope until it has processed all previous replies. Clearly this is efficient only in the case where the receivers are uniformly distributed in TTL bands, which may not be the case.

**Statistical probing.** This scheme relies on probabilistic arguments for scalability.

At the start of a round of probes (called *epoch*), the sender and each of the receivers generate a random key of a fixed bit length. In each probe, the source sends out its key together with a number specifying how many of the key digits are significant. Initially, all digits are significant. If a match occurs at a receiver then that receiver is allowed to send a response. If no response is received within a timeout period, the number of significant digits is decreased by one and another probe is sent. In [9], it was shown that there is a statistical relationship between the group size and the average round upon which a receiver first matches the key. This scheme is efficient in terms of number of replies needed to estimate the group size. However, as reported in [9], the maximum response time (the time needed for the source to identify the worst case of all receivers) is equal to 32 times the worst case round-trip time from the source to any of the group members. For a worst case RTT of 500 milliseconds, it may take up to 16 seconds to find the worst case state of all receivers.

**Randomly delayed replies.** In the randomly delayed replies scheme, each receiver delays the time at which it sends its response back to the source by some random amount of time. Clearly, the success of this scheme in preventing the *reply implosion* problem depends to a great extent on the duration of the period from which random delays are chosen. This scheme is appealing because it allows for receiving responses from all the receivers in the group, by adapting the delays using some knowledge of the size of the group.

From the above basic mechanisms, the *randomly delayed replies* approach, augmented with suppression of redundant replies and careful selection of delay periods, is the most appealing for two main reasons: first, a response is always guaranteed; and second, the response time is expected to be always low. This is the basic idea deployed in IGMP (Internet Group Management Protocol) [24]. In IGMP, the probe is sent to a single subnet, and hence as soon as one of the receivers responds to the probe it is guaranteed that all the other receivers will hear that response and suppress their replies, if necessary. Also, in such a local environment, the timeout period can be set to a fixed small value. In contrast, in our case, the group of receivers may be distributed over a wide area network (WAN), thus a reply sent by one receiver may not be heard by another before it emits its own reply which may be redundant. This implies the need for careful selection of the delay randomizing function.

A closely related, but different, problem is the negative acknowledgment (NAK) implosion problem associated with reliable multicast. A solution for the NAK implosion problem, which is based on randomly delayed replies with suppression of redundant NAKs, is adopted by the SRM protocol [34]. In SRM, when a receiver,  $i$ , detects a lost packet, it randomizes the delay before sending its NAK in the interval  $[C_1 d_i, (C_1 + C_2) d_i]$ , where  $d_i$  is the delay between receiver  $i$  and the source,  $C_1$  and  $C_2$  are constant parameters. Both of the NAK and state feedback implosion problems are similar in the need for soliciting replies from a potentially very large group of receivers. However, with NAKs, whenever a data packet is lost on a link, all the receivers that the faulty link lead to will eventually detect the loss and send a NAK. Thus the distance between a receiver and the faulty link is the major factor that determines when the receiver will detect the fault, and consequently favoring closer receivers, by letting them send their NAKs earlier, implies suppression of more redundant NAKs. On the other hand, in the state feedback problem, the capacity of the receiver, and consequently its state, may not be related to its distance from the source. Therefore, a different criteria for randomizing the delays is required.

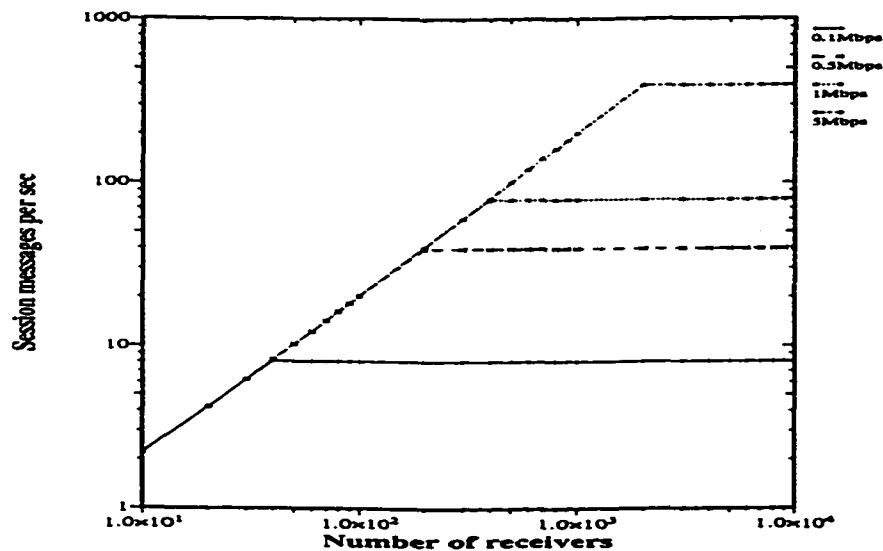


Fig. 2.1. Overhead of session messages.

In SRM, each receiver,  $i$ , must determine its distance,  $d_i$ , from the source to use it in the delay function. The overhead of session messages (typically RTCP reports [51]) which are needed for that purpose is not negligible. Figure 2.1, shows the overhead of RTCP reports for different session sizes and rates, assuming a single source. One of our objectives is to devise a scalable feedback mechanism that eliminates this high overhead, by designing the mechanism in a way that is not dependent on periodic session messages.

Another scalable feedback protocol, called SCUBA (scalable consensus-based bandwidth allocation), which also relies on periodic session reports, was recently introduced [4]. SCUBA is primarily concerned with scenarios where receivers have different views about the priorities of the streams in a session, and provides a scalable solution for averaging the priorities of the streams across receivers. These average weights can then be used in the bandwidth allocation process. Although the

statistical methods underlying the protocol are sound, the objective behind SCUBA is of minimal usefulness and applicability. The relative priorities of the different streams belonging to an IMC session are typically defined by the semantics of the application, which are identical at all receivers. Although these priorities vary dynamically, yet they vary consistently for all the session participants. However, one cannot completely rule out the possible utility of the protocol in other potential application domains.

In Chapter 4, we propose a new scalable and robust feedback protocol, which extends the randomly delayed replies approach, while avoiding the overhead of the periodic multicast of session messages.

# CHAPTER III

## INTER-STREAM BANDWIDTH ADAPTATION

In this chapter, we focus on the techniques used by the QoSess control layer to allocate the bandwidth available to an IMC application among the streams that compose together the IMC session. This allocation changes over time to match the dynamic nature of both the IMC streams and the network state. We address the issue of inter-stream bandwidth adaptation not only in the case of networks providing group reservation of resources, but in the case of networks providing only best-effort service as well. In the latter case, the service may be enhanced by a mechanism for differentiated services [28], but no guarantees are given.

The rest of this chapter is organized as follows. Section 3.1 presents the elements required for inter-stream bandwidth adaptation. Section 3.2 elaborates on one of these elements, which is an abstract method for representing the relative importance of the different streams to a session, in a way that isolates QoSess policies from mechanisms. In Section 3.3, the problem of inter-stream adaptation in presence of group reservation is abstracted as a simplified resource allocation model, and two strategies for resource allocation in a cooperative environment are devised and simulated. Section 3.4 presents a modified version of the linear bounded arrival processes (LBAP) model, M-LBAP, which is used to characterize traffic in a tighter way than the LBAP model while maintaining its simplicity. In Section 3.5 delay bounds are derived for traffic sharing the same group reservation and characterized

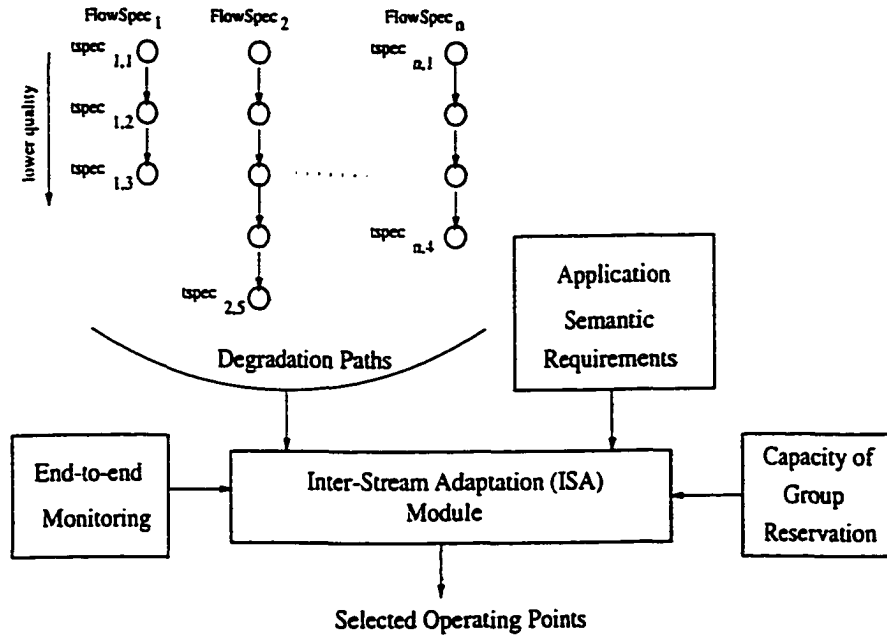


Fig. 3.1. Elements of inter-stream adaptation.

by the M-LBAP model. The two devised resource allocation strategies are revisited in order to support delay constraints and the resulting service is simulated. Section 3.6 addresses the issue of inter-stream adaptation in the absence of group reservation. Finally, the chapter is concluded in Section 3.7.

### 3.1 Elements of Inter-Stream Bandwidth Adaptation

The QoSess control layer allocates portions of the bandwidth available to an IMC session to the streams belonging to the session. As shown in Figure 3.1, an inter-stream adaptation (*ISA*) module uses information about the degradation paths of the streams belonging to the application, semantic requirements of the application, the capacity of group reservation if exists, and feedback information about the congestion state of the network, in the process of dividing the available bandwidth among the group of streams comprising the session, hence selecting the operating



point for each stream. Each of these vital elements to the inter-stream adaptation process is discussed in detail below.

### Degradation paths

Each operating point for a continuous media stream can be mapped from encoder specific parameters (e.g., frame rate, frame size, number of quantization levels, encoding technique used,...etc.) into traffic specific parameters. If each stream has one operating point, then no room is left for inter-stream adaptation besides turning on/off some of the streams based on the availability of resources and the set of simultaneously active streams. On the other hand, arranging more than one operating point for a stream in the form of a degradation path, where each node in the path represents a lower level of quality of service with respect to the previous node, gives more flexibility for the ISA module in adapting to availability of resources or changes in application requirements.

The flow specification (*FlowSpec*) of a stream is composed of traffic specification (*TSpec*) and QoS requirements specification (*RSpec*). The *TSpec* represents an ordered list of operating points:  $TSpec = (tspec_1, tspec_2, \dots, tspec_m)$ . The *RSpec* represents the delay, jitter, and loss constraints for the stream as well as the relative importance of each of these factors. The rate requirements are implicitly specified by the selected operating point. Many flow specification models have been proposed in the literature. These mainly differ in the way of characterizing the traffic. In Section 3.4, we describe several generic traffic characterization models.

The *FlowSpec* of a stream defines a degradation path, as shown in Figure 3.1, where the head node in the path represents the most preferred operating point from the user perspective, and the tail node of the path represents the least acceptable operating point for that stream.

### **Application semantic requirements**

In an IMC application, the group of streams belonging to the application have a highly dynamic nature. A stream may be started/stopped at any instant. Moreover, the relative priority of a stream with respect to the other streams varies over time. The priority of a stream is a measure of the contribution of that stream to the total view presented to the users. The priority of a stream at any instant in time is a function of the set of active streams at that instant. In Section 3.2, we present a general graph abstraction for representing the relative importance of the different streams to the application semantics.

In addition to priorities, other types of relationships between groups of streams may be implied by the semantics of the application. For example, a pair of streams may be required to be always in the same active/inactive state, e.g., audio and video from the same source. Considerations for semantic requirements of the application are scoped, in this dissertation, to capturing the relative priorities of the streams and reflecting these priorities on the inter-stream adaptation decisions.

### **Capacity of group reservation**

In a networking environment where group reservation is provided to applications in order to support sharing of resources, the QoSess control layer must allocate fractions of the total amount of reserved bandwidth to each stream. Therefore, an important factor that the ISA module has to consider while making resource allocation decisions is the total amount of resources dedicated to the streams of the application. Even, if group reservation support is not provided, knowledge about the maximum capacity of the shared resource (e.g., the bandwidth of the Internet connection to a home user) is useful in preventing network overload due to decisions made by the QoSess layer.

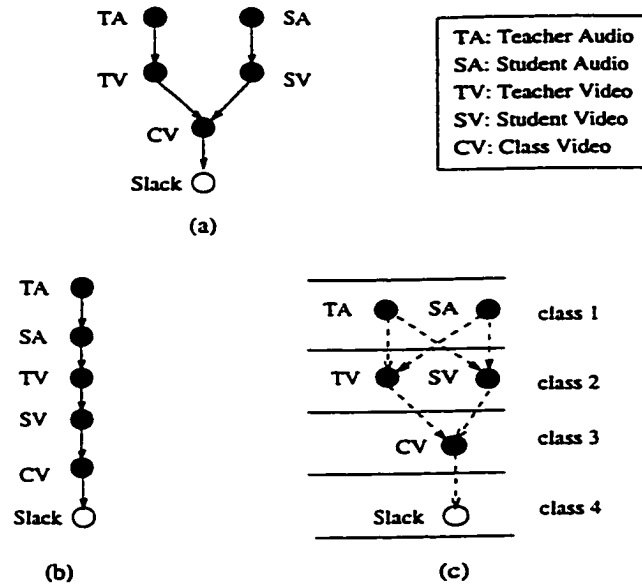


Fig. 3.2. Representing the relative priorities of streams. (a) QoSess graph. (b) Priority list. (c) Priority classes.

### End-to-end monitoring

In the absence of group reservation, the QoSess layer relies on capacity estimation techniques in order to determine the total bandwidth available to the session. This is done by active probing of the network, through activating as many streams as possible until congestion signals are detected by the end-to-end monitoring component of the QoSess layer. A congestion signal can be in the form of detection of packet losses, an increase in delay variation or other explicit signals that the network layer may provide.

Monitoring the overall end-to-end performance is not only important for best effort systems, but for systems that provide statistical or predicted guarantees as well, in order to react to potential overload situations.

## 3.2 The Quality of Session Graph

In order to represent the relative priorities of the streams, as dictated by the application semantics, we devised a graph representation, called the *QoSess graph*. A QoSess graph is a direct acyclic graph  $G(V, E)$ , where  $V$  is the set of nodes in the graph; and  $E$  is the set of edges. Each node corresponds to one active stream. An edge directed from node  $i$  to node  $j$  means that stream  $i$  has higher priority than  $j$  at this instant in time. Consequently, adaptation decisions allow  $i$  to borrow resources from  $j$ . An additional node, called the *Slack* node is added such that every stream can borrow from it. Hence, this representation defines for each node (stream) the set of nodes from which it can borrow resources. We call this set the *borrow set*. The borrow set  $B_k$  for stream  $k$ , can be recursively defined as:

$$B_k = \bigcup_{(k,j) \in E} (\{j\} \cup B_j)$$

For example, Figure 3.2(a) depicts the QoSess graph representing the relative priorities of a set of streams that are typical in the IRI (Interactive Remote Instruction) system [45].

The QoSess graph is a general and flexible representation for the relative priorities of the streams belonging to an application: the borrow set can be defined under this representation with great flexibility; and other common representations for relative priorities can be considered as special cases of the QoSess graph, as in Figures 3.2(b) and (c).

### 3.2.1 Mapping graphs to priority classes

The complexity of the inter-stream adaptation mechanism is affected by the method by which the relative priorities of streams are represented. An inter-stream adaptation algorithm based on the general QoSess graph representation is the most complex, but has the advantage of giving maximum flexibility to the application in defining the borrow sets of the streams. A compromise between flexibility and com-

plexity may be achieved using priority classes. In order to combine the semantic flexibility of the QoSess graph with the relatively simple algorithmic management of priority classes, we devised a mapping algorithm which maps a QoSess graph to a corresponding set of priority classes.

This mapping algorithm decouples the inter-stream adaptation policy from the mechanism used to implement it. The application specifies its needs in a flexible way that reflects its semantic requirements without interfering with the parameters (priority weights) that control the adaptation process.

Invoking the two-step algorithm *G2P*, which is depicted in Figure 3.3, results in the computation of the depths of all nodes in  $G$  in a single depth-first traversal of the graph. Figure 3.4(a) depicts an example QoSess graph together with the computed node depths. The priority of each node  $k$ ,  $p_k$ , is then set to  $d_{slack} - d_k$ , where  $d_k$  is the depth of node  $k$ . This is illustrated in Figure 3.4(b). This two-step process is necessary to avoid excessively low priority assignments to shallow partitions of the graph.

Practical situations where the QoSess graph could be partitioned in a way similar to Figure 3.4 are likely to happen, e.g., a group of students may start a collaborative distance learning session that employs multiple multimedia streams, and decide to perform a group review of a previous session. Another example is a group of scientists collaborating using a multimedia session in which they trace and discuss the progress of a distributed simulation system which in turn uses multiple streams for data and visualization. In each of the above examples, two independent applications, each deploying multiple streams, run concurrently and compose one session. Using this mapping algorithm, the QoSess layer is able to make the two applications cooperate together to present the user with the best session quality, within the limits of the available resources, even though each of the two applications has no information about the streams deployed by the other.

The time complexity of *G2P* is the same as the depth-first search traversal complexity which is  $O(n+e)$ , where  $n$  is the number of nodes, and  $e$  is the number of

```

G2P()  {
    //Step 1: Compute depths
    Compute_Depth(slack) ;
    //Step 2: Assign priorities
    for-each node  $k$ 
         $p_k = d_{slack} - d_k$  ;
    }

    Compute_Depth( $k$ )  {
        //All  $d_i$  are assumed to be initially set to 0
        for-each node  $i$  such that node  $i$  is a parent of node  $k$  {
            if (  $d_i \neq 0$  ) //already visited node  $i$ 
                 $d = d_i$  ;
            else          //visit node  $i$  now
                 $d = \text{Compute\_Depth}(i)$  ;
            if (  $d > d_k$  )
                 $d_k = d$  ;
        }
         $d_k = d_k + 1$  ;
        return  $d_k$  ;
    }
}

```

Fig. 3.3. The G2P mapping algorithm.



or alternatively the client's request could be rejected if the resource manager cannot reach an agreement with the client about the value of the allocation. Recently, this approach has been followed in many resource allocation and reservation systems for the support of multimedia streams in distributed environments [6, 14, 18].

While this approach has proven to be effective for handling individual independent connections, and guaranteeing a stable performance for the admitted connections, there are situations in which strict admission tests and fixed allocations of resources over the lifetime of a connection is not suitable: it leads to under utilization of resources; and provides a lower quality of service to the distributed application. Consider the simple example of an audio conferencing tool. If resources are reserved for each potential speaker, while the application semantics imply that only one participant can speak at a time, we end up wasting the majority of the reserved resources. If on the other hand a connection is established/torn-down on the fly whenever a speaker starts/stops speaking, we face the potential of one or more of the connections being rejected as well as the latency involved in the connection establishment time.

Indeed, the insufficiency of this model, of fixed resource allocations to individual streams, for handling situations where multiple streams cooperate to compose the view presented to the user was recognized, and several proposals were made to augment it by supporting group reservation [7, 67]. The idea behind group reservation is to allow a group of streams belonging to the same distributed application to share a common repository of resources. This common set of resources represents the fixed allocation that is guaranteed by the system. In this case, it is desired to maximize the overall benefit gained by the distributed application from the shared resources, rather than solely focusing on guaranteeing performance for the already admitted connections. This may imply temporary shut-down of an active connection to allow a more important connection to use the shared resources, if this yields a better overall gain to the performance of the distributed application.

We abstract the problem in terms of a resource manager (RM) that owns



a shared resource, and allocates fractions of that resource to clients according to a specific policy. Each client has the ability to adapt dynamically to the allocated level of the resource within a bounded range. In addition, the resource manager may choose to temporarily preempt some of the clients for the sake of better overall performance of the distributed application. Our objective is to devise allocation policies which are suitable for the friendly environment and cooperative nature of the streams belonging to an IMC session. We focus on the effect of application of different allocation policies, under the assumption of the existence of suitable mechanisms for enforcing the shares allocated by these policies, e.g., a scheduler in the case of a time-shared resource. Alternatively, in this friendly environment, it can be safely assumed that each client will not exceed its share of the resource, hence scheduling and policing techniques are not a necessity. The abstract model simplifies the problem by isolating it from the details of a particular application domain, and generalizes it in order to allow for investigating its applicability in more than one domain.

### 3.3.1 Resource allocation models

The two generic resource allocation models, namely the *fixed-point* model and the *range-based* model, are described here in more detail.

In the *fixed-point* allocation model, the client specifies its requirement as a fixed level of the resource,  $R_{fixed}$ . This is in contrast to the range of operation,  $[R_{min}, R_{max}]$ , in the *range-based* model. If the resource level allocated to the client drops below  $R_{fixed}$ , the client is considered to be preempted temporarily. Also, the client cannot adapt itself to benefit from any resource allocation above  $R_{fixed}$ . In addition, a priority level is associated with each client.

It should be noted that some systems allow the client to specify a range of requirements that is considered only in the initial negotiation phase. As soon as a certain level is agreed upon, it is fixed throughout the lifetime of the client.

We classify this scheme under the fixed-point model and the agreed upon level is considered as the fixed-point client requirement throughout the session.

The fixed-point resource allocation model is the commonly used model in reservation systems [6, 14, 67]. It is typically applied in conjunction with a non-preemptive allocation policy, where an admitted client is guaranteed its allocation throughout its lifetime. The fixed-point model can be viewed as the special case  $[R_{fixed}, R_{fixed}]$  of the range-based model.

### Client model

Clients arrive at the RM at any instant in time. Associated with each client are:

1. a priority level  $p$ ; and
2. a resource requirement, which depending on the model used can be represented by either  $R_{fixed}$ , or a range  $[R_{min}, R_{max}]$ , of fractions of the total amount of the resource.

In the range-based model, as long as the client does not leave the system, it is capable of benefiting from any resource allocation up to  $R_{max}$ , and it can dynamically adapt itself according to the availability of the resource at any level in the range  $[R_{min}, R_{max}]$ . If the level of the resource allocation given to the client drops below  $R_{min}$ , the client is considered to be preempted temporarily.

### Cooperating versus competing clients

Clients may have a unified goal, e.g., all the streams of a distributed multimedia system cooperate together in order to present an integrated view to the user. These cooperating clients are assumed to be friendly to each other. In terms of resource allocation, this implies that resources may be taken from one client to be given to another provided that this helps the overall unified goal.

In other situations, this may not be the case, and hence we refer to the clients in that case as competing clients. The cooperative or competitive nature

of the clients affects the policy that should be followed by the RM in allocating resources.

### **Dynamic priorities**

The priority associated with the client is allowed to change over time. This triggers the RM to check all the allocations and take any corrective decisions regarding the current allocations if necessary.

### **Preemption versus non-preemption**

In the range-based model, a client is considered to be preempted if the level of the resource allocation given to the client drops below  $Rmin$ . In this case any amount of resource given to the client is wasted as the client cannot benefit from it, so a zero allocation should be given in this case.

In a preemptive version of the range-based model, a client can be temporarily preempted, by the RM, by allocating a zero level of the resource to it. In a non-preemptive version, once a client is admitted, the system is committed not to decrease the resource allocation to that client below  $Rmin$ , throughout the lifetime of the client. In what follows, we consider only the preemptive version, since it lends itself to the cooperative environment, while the non-preemptive version matches the competitive mind-set.

### **3.3.2 RISA: An optimizing approach for range-based resource allocation**

*RISA* stands for *Rate-based Inter-Stream Adaptation*. The naming of the scheme was influenced by the application of interest to us, which is controlling the rates of several cooperating streams in a distributed multimedia session. In RISA, resource allocation is done in two phases as follows.

1. **Selection Phase.** In this phase, a subgroup of the clients are selected to be granted access to the resource. Each of these is given its required  $R_{min}$  level of the resource. The selection process depends on the priorities of the clients and the total amount of the available resource.
2. **Enhancement Phase.** In this phase, the remaining non-allocated amount of the resource is divided among the selected clients. The objective is to make the share of each active client as close as possible to its specified  $R_{max}$ , while maximizing the overall benefit gained by the group of clients from this allocation.

The selection and enhancement phases are executed whenever a client arrives/departs or a change in priority occurs. These two phases are detailed below.

### The selection phase

All clients are scanned in descending order of priority, granting each the requested  $R_{min}$  level, until either the total amount available of the resource is exhausted or all the clients are examined.

In the non-preemptive case, this phase is somehow different. Instead of selecting a sub-group of clients from all those that are in the system to grant access to the resources, those that are already granted access are always guaranteed to have their  $R_{min}$  requirement. At the arrival of a new client to the system, an admission test is made to see if  $R_{min}$  of that client can be granted while satisfying the minimum requirements of all the already active clients. If it can be granted then the client becomes active, otherwise it is kept passive and the system tries to admit it whenever an active client leaves the system, until it is either admitted or decides to leave the system.

### The enhancement phase

After allocating the basic needs of the selected clients, the enhancement phase allocates the remaining resources to the selected clients with the objective of maximizing the overall gain to the system from this allocation.

In the approach taken here, this resource allocation problem is formulated as an optimization problem that reduces to the continuous form of the well known knapsack problem [19].

$$\text{Maximize } \sum_{i=1}^n (p_i * f_i)$$

subject to:

$$\sum_{i=1}^n [f_i * (Rmax_i - Rmin_i)] \leq 1 - \sum_{i=1}^n Rmin_i$$

$$0 \leq f_i \leq 1 \quad \text{for } i = 1, 2, \dots, n$$

where,

$n$  is the number of clients selected in phase 1;

$p_i$  is the priority of client  $i$ ;

$Rmin_i$  is the minimum requirement of client  $i$ ,  $0 \leq Rmin_i \leq 1$ ;

$Rmax_i$  is the maximum requirement of client  $i$ ,  $0 \leq Rmax_i \leq 1$ ;

$f_i$  is the fraction of  $(Rmax_i - Rmin_i)$  which is granted to client  $i$ .

At the end of the two phases, if client  $i$  is active then it is assigned  $Rmin_i + f_i(Rmax_i - Rmin_i)$ , otherwise, if it is not one of the selected active clients then no resources are assigned to it.

The continuous form of the knapsack problem is a special linear optimization problem for which an optimal solution can be obtained by traversing the list of clients in the order of  $p_i/(Rmax_i - Rmin_i)$  and giving each client its maximum need until all resources are exhausted [19]. In our case, the cost of sorting can be avoided by maintaining all client requests in a list sorted according to  $p_i/(Rmax_i - Rmin_i)$ , hence reducing the time complexity of the algorithm from  $O(n \log n)$  to  $O(n)$ .

### 3.3.3 I-WFS: A fair approach for range-based resource allocation

In spite of the fact that RISA generates the optimum allocation for the enhancement phase, it has an inherent characteristic that may be considered as a deficiency in some systems: it does not attempt to achieve fairness. It is worth mentioning that, in many systems, this may not be considered as a drawback at all. In particular, in systems of cooperating clients with global objectives, optimality of resource allocation may be considered as far more important than fairness in the process of resource allocation. Nevertheless, for many systems fairness is a critical issue. For such systems, we propose a modified version of the *Weighted Fair Share (WFS)* allocation strategy [13], called *Iterative Weighted Fair Share (I-WFS)*. In WFS, client  $i$  has a weight,  $w_i$ , associated with it, and is granted an amount of the resource proportionate to  $w_i$ , assuming that each client can accept any allocation ranging from null to the maximum available amount of the resource which is typical in best-effort systems.

In I-WFS, the priority of the client,  $p_i$ , is used as the distinguishing factor between clients, and the weight of client  $i$  is  $w_i = \frac{p_i}{\sum_{j=1}^n p_j}$ . In addition, the client requirements are defined by the range  $[Rmin_i, Rmax_i]$ . This requires modification of the original scheme. We follow the same approach taken in RISA by dividing the task into two phases.

1. **Selection Phase.** In this phase, a subgroup of the clients is selected to be granted access to the resource. Each of these is given its required  $Rmin$  level of the resource. The selection process depends on the priorities of the clients and the total amount of the available resource. The non-preemptive case is handled in the same way as it is handled in RISA.
2. **Enhancement Phase.** In this phase, the remaining non-allocated amount of the resource is divided among the selected clients in an iterative method. In each iteration, the weight of each active unsatisfied client is computed, as

above, but relative only to the other active unsatisfied clients. The available amount of the resource is tentatively divided according to the computed weights and the share of each client is checked against its remaining need, to reach  $Rmax_i$ . If the share is less than the remaining need, it is granted to the client and the client is marked as still unsatisfied. Otherwise, the client is given only its remaining need. This process is iterated until either all the clients are satisfied, or all the resources are allocated.

In the worst case,  $n$  iterations are needed until all the clients are satisfied. In each iteration, the weights of the unsatisfied clients are computed, and only one client is satisfied. Hence, the worst case time complexity of the algorithm is bounded by  $O(n^2)$ .

### 3.3.4 A metric for comparing resource allocation policies

In order to compare the behavior of the system under the application of different resource allocation strategies, we propose a unified metric that reflects the overall performance of the system for a given allocation, from the clients perspective. We define  $Q_i$  as the degree of satisfaction of client  $i$ . The *aggregate satisfaction level* for all the clients is obtained by computing the weighted arithmetic mean of the set of  $Q_i$  values for all  $i$ , and is used to represent the quality of session (QoSess).

$$Q_i = \begin{cases} R_i/Rmax_i & \text{if } i \text{ is active} \\ -1 & \text{if } i \text{ is not active} \end{cases} ;$$

$$QoSess = \frac{\sum_{i=1}^n (p_i * Q_i)}{\sum_{j=1}^n p_j} ;$$

where,  $R_i$  is the current amount of resource allocated to client  $i$ . The system is penalized by -1 for each inactivated stream. The value of QoSess lies in the interval  $[-1,1]$ . The best attainable value for QoSess may be below one sometimes. This is not a concern because the QoSess metric is intended for comparing different allocation policies relative to each other.

Accounting for this client oriented metric, besides other typical system oriented metrics such as utilization and acceptance ratio, yields a better understanding of the performance of the different allocation strategies.

### 3.3.5 Preliminary simulation results

We have implemented the range-based resource allocation policies RISA and I-WFS, as well as a fixed-point resource manager for the purpose of simulating and contrasting the performance of the different resource allocation models and policies. In the simulation experiments, an input event may belong to one of the following three types of events.

1. **Request.** A new client arriving at the system.  $R_{min}$ ,  $R_{max}$  and  $p$  are specified. For the fixed-point case only one value,  $R_{fixed}$ , of resource requirement is specified together with  $p$ .
2. **Release.** A client leaving the system.
3. **Change priority.** A client announces its new priority level.

Two kinds of input sequences of events were used to drive the simulations: randomly generated events; and synthesized scenarios. With randomly generated events, clients are assumed to arrive at the system according to a poisson distribution. The client remains in the system for a period of time that is generated according to an exponential distribution. In the case of synthesized scenarios, a sequence of input events that represents a real-world or a hypothesized scenario in a particular system, is used.

For comparing the fixed-point model to the range-based model, three fixed-point policies were considered:

1. **min**, where  $R_{fixed}=R_{min}$ ;
2. **max**, where  $R_{fixed}=R_{max}$ ; and



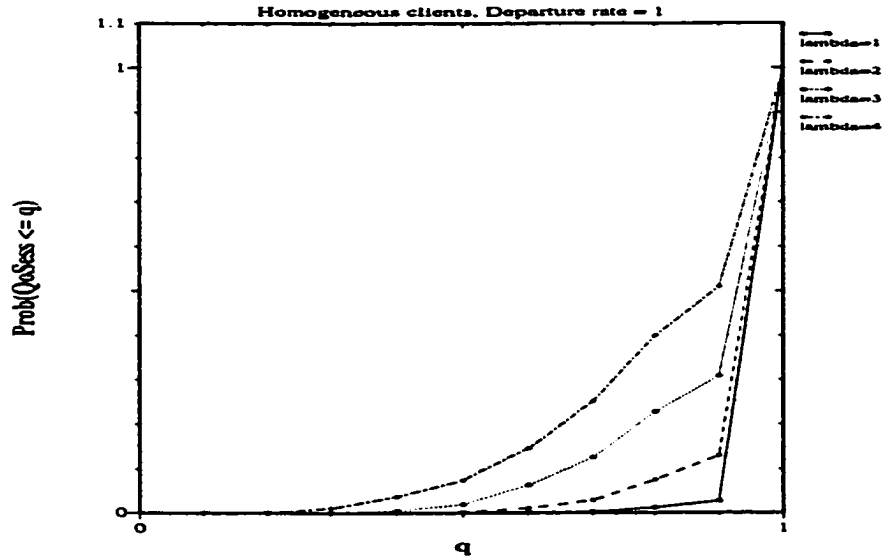


Fig. 3.5. QoSess CDF variation with load for RISA.

3. avg, where  $R_{\text{fixed}} = (R_{\text{min}} + R_{\text{max}}) / 2$ .

These three policies cover all the extremes. The first case represents a conservative group of clients, or a system that is designed for worst case scenarios. The second case represents an aggressive group of clients, and in between lies the average case.

Figures 3.5 and 3.6 show the variation in the cumulative distribution function (CDF) of the QoSess metric with the offered load. The offered load is varied by changing the arrival rate ( $\lambda$ ) of the clients. The randomly generated clients are homogeneous, with resource range  $[0.1, 0.3]$  for RISA and random integer priority in the range  $[1, 5]$ . For the fixed-point case, clients request the average value which is 0.2. The figures show the behavior of the system from start time until the client number 1000 leaves the system (the simulated time varies with arrival rate). From the figures, it is clear that the rate of degradation in QoSess with the increase in load is much lower in the range-based case than in the fixed-point case.

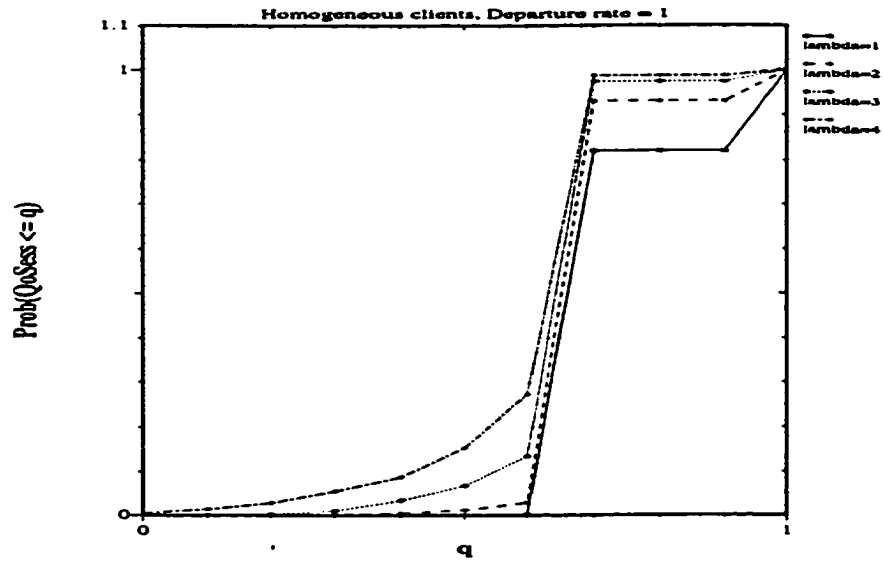


Fig. 3.6. QoSess CDF variation with load for average fixed-point.

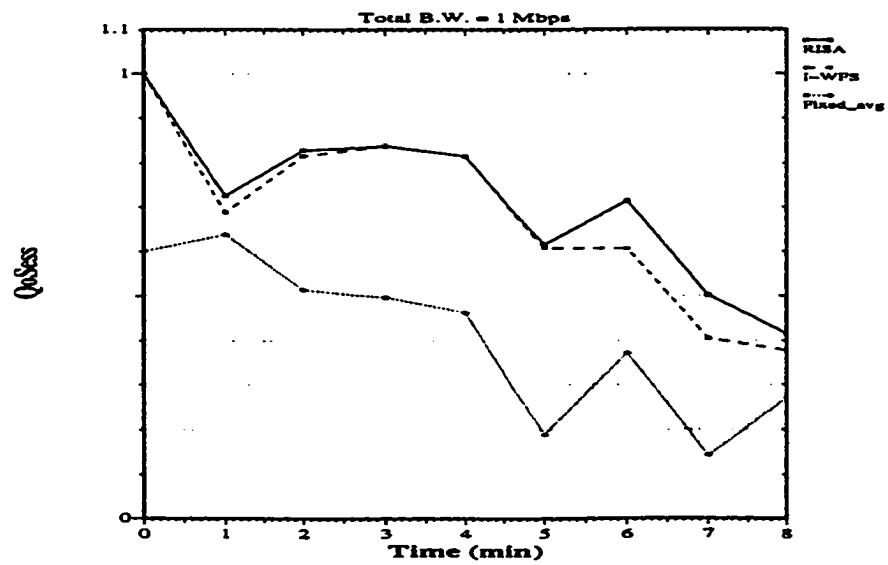


Fig. 3.7. QoSess in an IRI derived scenario.

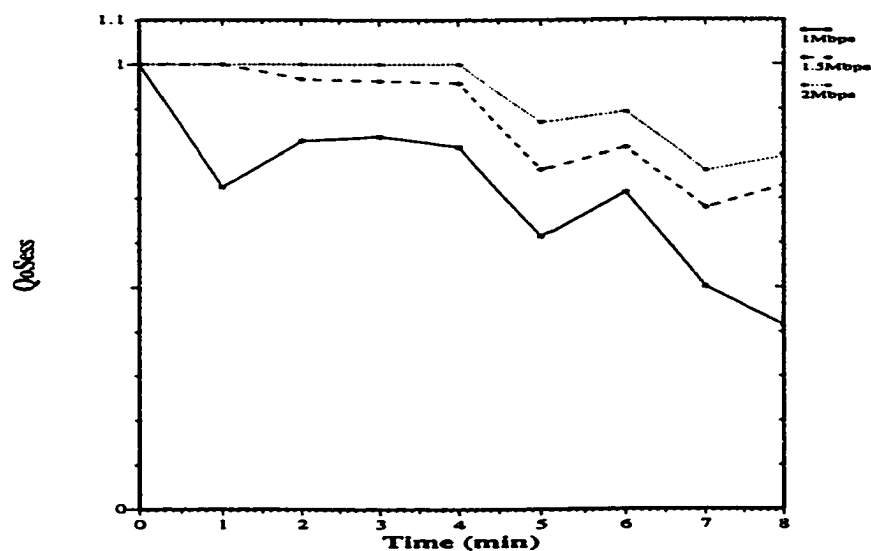


Fig. 3.8. Effect of resource availability on QoSess for RISA.

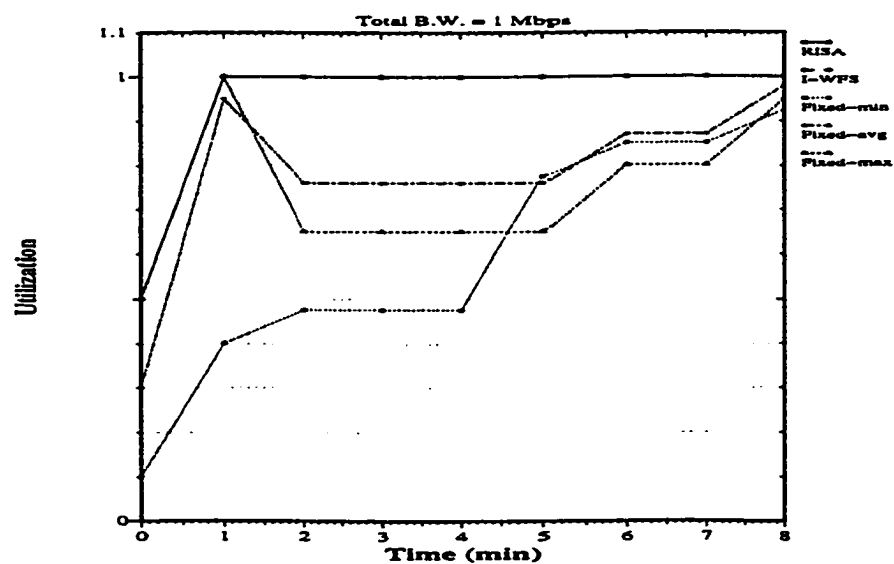


Fig. 3.9. Resource utilization in an IRI derived scenario.

Figures 3.7 through 3.9 represent the results of running the simulations with an input sequence that represents a real-world scenario derived from the IRI system. The scenario represents a sequence of start/stop times of four video streams and three audio streams that share the same network bandwidth resource. Figure 3.7 emphasizes the superiority of the range-based approach over the fixed-point approach, as the QoS<sub>sess</sub> obtained by RISA represents an upper bound for all other strategies at all instants of time. Also from the figure, we see that I-WFS yields QoS<sub>sess</sub> values that are very close to RISA, although RISA is always at equal or higher level. The high QoS<sub>sess</sub> values obtained by I-WFS make it a good candidate for the range-based model as it achieves fairness in the allocation of the enhancement layers. RISA on the contrary strives only to optimize the QoS<sub>sess</sub> without any attempt to be fair in the resource allocation.

Figure 3.8 shows how the QoS<sub>sess</sub> can be enhanced if more resources are made available. This is true for fixed-point as well as range-based policies.

Figure 3.9 demonstrates that both RISA and I-WFS are capable of better utilizing the available resources always, while in the fixed-point strategies resources may be wasted. In this IRI scenario, the fixed-point strategies start achieving high utilization values close to those of RISA and I-WFS, when the maximum load offered increases to almost four times the available resources.

From the presented graphs, we can deduce that the range-based model of resource allocation is more suitable than the fixed-point model, for groups of cooperating clients with a unified goal. Better utilization of resources and satisfaction of clients requirements are achieved by the range-based model. An important conclusion also is that I-WFS performs very close to RISA, besides its own advantage of being fair in allocation of resources.

In the following sections, we further explore the performance of the range-based resource allocation model in IMC sessions that impose delay constraints on the packets of the streams composing a session.

### 3.4 Traffic Characterization

A key issue for providing quality of service guarantees is the ability to characterize the traffic of each stream for which guarantees are being provided. A traffic characterization model must be tight enough to avoid excessive allocation of resources, and simple enough for the application to use in its specification and for the network to be able to support, as well as for the analysis to be tractable. In addition, the model should allow for the aggregate characterization of the traffic of a group of streams sharing the same resources which are reserved for the group.

Cruz [22] developed bounding techniques based on a fluid traffic model  $(\sigma, \rho)$ . Central to the analysis is the concept of *traffic constraint function*  $b(t)$ .  $b(t)$  is defined to be the maximum number of bits that can arrive during any interval of length  $t$ . For the  $(\sigma, \rho)$  model,  $b(t) = \sigma + \rho t$ .

The linear bounded arrival processes (LBAP) model [5, 53], characterizes the traffic using three parameters  $(R, B, S)$ , where  $R$  is the average rate in bits/sec,  $B$  is the maximum burst size in packets, and  $S$  is the maximum packet size in bits. It can be easily shown that the LBAP model is simply a  $(\sigma, \rho)$  model with  $\sigma = BS$  and  $\rho = R$ . The LBAP model has the advantage of being simple for the application to use in its specification as well as for the network to use in its implementation in order to support the specified stream characteristics.

Ferrari et al. [31], use the discrete model  $(X_{min}, X_{ave}, I, S)$ , where  $X_{min}$  is the minimum packet inter-arrival time,  $X_{ave}$  is the average packet inter-arrival time,  $I$  is the averaging interval, and  $S$  is the maximum packet size. In [69], the bounding function  $b(t)$  for the discrete model is given by  $\left( \min \left( \left\lceil \frac{t \bmod I}{X_{min}} \right\rceil, \left\lceil \frac{I}{X_{ave}} \right\rceil + \left\lceil \frac{t}{I} \right\rceil \left\lceil \frac{I}{X_{ave}} \right\rceil \right) \right) S$ . The discrete model is tighter in characterizing streams but lacks a lot of the simplicity of the LBAP model. Also, determining the optimum value of  $I$  is not a trivial task and may be impossible for real-time traffic.

The model we use is derived from the LBAP model. It strikes a balance between the simplicity of specification and analysis of the LBAP model and the

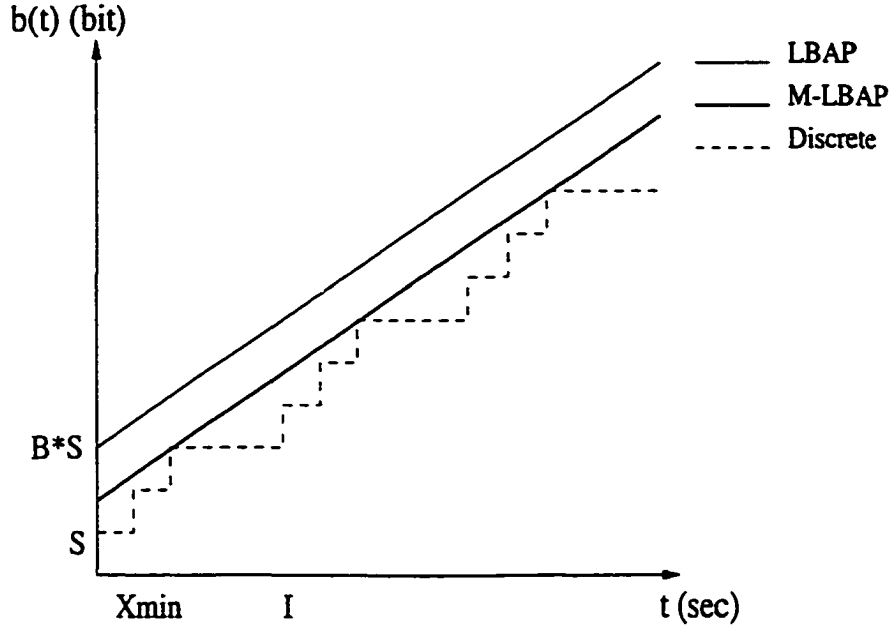


Fig. 3.10. Bounding functions of three traffic characterization models.

accuracy of representation of the discrete model. We call it the *Modified-LBAP* (*M-LBAP*) model.

In M-LBAP, a stream is characterized by four parameters  $(R, B, S, PAR)$ , where the first three parameters are the same as the LBAP original parameters, and  $PAR$  is the rate peak-to-average ratio or the burst ratio. Figure 3.10 shows a graphical representation for the bounding functions of the different models. It can be easily shown that for M-LBAP, the bounding function  $b(t)$  is given by  $BS \left(1 - \frac{1}{PAR} \left(1 - \frac{1}{B}\right)\right) + Rt$

M-LBAP, is also a  $(\sigma, \rho)$  model with  $\sigma = BS \left(1 - \frac{1}{PAR} \left(1 - \frac{1}{B}\right)\right)$  and  $\rho = R$ . This model provides a tighter characterization for the burstiness of a stream than the LBAP model and hence avoids the excessive allocation of resources.

One of the main advantages of having a linear model derived from the  $(\sigma, \rho)$  model is the ability to characterize a group of streams, as a single aggregate stream. In [68], it was shown that the aggregate traffic of  $K$  streams, each satisfying  $(\sigma_k, \rho_k)$ ,  $k = 1, 2, \dots, K$ , satisfies  $(\sum_{k=1}^K \sigma_k, \sum_{k=1}^K \rho_k)$ . This characteristic of the M-LBAP model

makes it adequate for characterizing the streams sharing a group reservation, and regarded by the underlying network as a single aggregate stream.

## 3.5 Bounding Delays

Using a tight traffic characterization model, that accurately captures the source behavior, allows for computing delay bounds for the streams and supporting the end-to-end delay requirements of the users. In a packet-switching network, the end-to-end delay of a packet consists of the following four components [68].

1. Link delay, which includes the propagation delay and other delays incurred in intermediate subnetworks if some of the links are subnetworks.
2. Switching delay, which depends on the implementation of the switches.
3. Transmission delay, which is a function of the packet length and link speed.
4. Queuing delay at each switch.

Under the assumption that there are no intermediate subnetworks, or alternatively that all intermediate nodes have reservation capabilities, the link delay is constant and equal to the propagation delay. The switching delay is fixed. Knowing the link speed and the maximum packet length makes the transmission delay fixed as well. The queuing delay is the component that can be affected by controlling the load or using an appropriate service discipline, and hence is the major concern.

### 3.5.1 Bounding delays in a FCFS scheduler

The following theorem was stated and proven in [69].

**Theorem 1:** *Let there be  $n$  channels multiplexed on a link with a FCFS scheduler and link speed  $l$ . If for  $j = 1, \dots, n$ , the traffic on channel  $j$  is bounded by  $b(\cdot)$ , then*

the delays of packets on all the channels are bounded by  $d$ , where  $d$  is defined by

$$d = \frac{1}{l} \max_{u \geq 0} \left\{ \sum_{j=1}^n b_j(u) - lu \right\} + \frac{S_{max}}{l} ,$$

where,  $S_{max}$  is the maximum packet size that can be transmitted over the link.

Including  $\frac{S_{max}}{l}$  accounts for the fact that a lower priority, non-real time, packet may be in transmission and cannot be preempted.

The following theorem builds on Theorem 1 to define the delay bounds for a FCFS scheduler and a group of streams whose traffic obey the M-LBAP model.

**Theorem 2:** Let there be  $n$  channels multiplexed on a link with a FCFS scheduler and link speed  $l$ . If for  $j = 1, \dots, n$ , the traffic on channel  $j$  obeys the M-LBAP traffic specification  $(R_j, B_j, S_j, PAR_j)$ , and if  $\sum_{j=1}^n R_j \leq l$ , then the delays of packets on all the channels are bounded by  $d$ , where  $d$  is defined by

$$d = \frac{1}{l} \left\{ \sum_{j=1}^n B_j S_j \left( 1 - \frac{1}{PAR_j} \left( 1 - \frac{1}{B_j} \right) \right) \right\} + \frac{S_{max}}{l} ,$$

where,  $S_{max}$  is the maximum packet size that can be transmitted over the link.

### Proof

From Theorem 1,

$$\begin{aligned} d &= \frac{1}{l} \max_{u \geq 0} \left\{ \sum_{j=1}^n b_j(u) - lu \right\} + \frac{S_{max}}{l} \\ &= \frac{1}{l} \max_{u \geq 0} \left\{ \sum_{j=1}^n B_j S_j \left( 1 - \frac{1}{PAR_j} \left( 1 - \frac{1}{B_j} \right) \right) + R_j u - lu \right\} + \frac{S_{max}}{l} . \end{aligned}$$

Since  $\sum_{j=1}^n R_j u \leq lu$ ,

therefore,

$$d \leq \frac{1}{l} \left\{ \sum_{j=1}^n B_j S_j \left( 1 - \frac{1}{PAR_j} \left( 1 - \frac{1}{B_j} \right) \right) \right\} + \frac{S_{max}}{l} .$$



The next two corollaries follow from Theorem 2. They define the delay bounds for a group of streams that share a fraction of the total transmission rate of a link.

**Corollary 1:** *Let there be  $n$  channels sharing a group reservation at the rate of  $R_{tot}$ , on a link with a non-work conserving scheduler and link speed  $l$ . If for  $j = 1, \dots, n$ , the traffic on channel  $j$  obeys the M-LBAP traffic specification  $(R_j, B_j, S_j, PAR_j)$ , and if  $\sum_{j=1}^n R_j \leq R_{tot}$ , then the delays of packets on all the channels are bounded by  $d$ , where  $d$  is defined by*

$$d = \frac{1}{R_{tot}} \left\{ \sum_{j=1}^n B_j S_j \left( 1 - \frac{1}{PAR_j} \left( 1 - \frac{1}{B_j} \right) \right) \right\} + \frac{S_{max}}{l},$$

where,  $S_{max}$  is the maximum packet size that can be transmitted over the link.

#### Proof

A non-work conserving scheduler will always serve the group of channels at the rate of  $R_{tot}$ , even if its capacity is higher and all the other channels are idle. Also, it is implicitly assumed that the scheduler will serve packets belonging to the group in the order of their arrival. Given these two facts, the problem in hand reduces to that of Theorem 2, with the exception that the service rate for the group of channels is  $R_{tot}$  instead of  $l$ .

**Corollary 2:** *Let there be  $n$  channels sharing a group reservation at the rate of  $R_{tot}$ , on a link with a work conserving scheduler and link speed  $l$ . If for  $j = 1, \dots, n$ , the traffic on channel  $j$  obeys the M-LBAP traffic specification  $(R_j, B_j, S_j, PAR_j)$ , and if  $\sum_{j=1}^n R_j \leq R_{tot}$ , then the delays of packets on all the channels are bounded by  $d$ , where  $d$  is defined by*

$$d = \frac{1}{R_{tot}} \left\{ \sum_{j=1}^n B_j S_j \left( 1 - \frac{1}{PAR_j} \left( 1 - \frac{1}{B_j} \right) \right) \right\} + \frac{S_{max}}{l},$$

where,  $S_{max}$  is the maximum packet size that can be transmitted over the link.

### Proof

A work conserving scheduler will serve the group of channels at least at the rate of  $R_{tot}$ . If its capacity is higher and some of the other channels are idle, the scheduler may sometimes serve the group of channels at a rate higher than  $R_{tot}$ . Given this fact, the problem in hand reduces to that of Corollary 1, with the exception that the delay bound here is less tight.

### 3.5.2 RISA and I-WFS revisited

The two algorithms, RISA and I-WFS, need to be revised in order to support the delay bounds specified in the flow specification of the streams. In addition, it is required to demonstrate the applicability of the algorithms for different traffic characterization models. This is achieved by examining the modifications needed to support the M-LBAP traffic model. In this case, we consider a delay bound constraint that must be respected.

The degradation path here assumes continuous values in the range  $[R_{min}, R_{max}]$  for the parameter  $R$  of the M-LBAP model, while the other parameters are fixed. This is equivalent to changing only the sampling rate (frame rate for video) of the encoder while keeping all other precision and quality parameters of the encoder constant.

The only modification needed for the two algorithms is in the selection phase. The enhancement phase remains as previously specified for each. In the selection phase, all streams are scanned in descending order of priority, granting each its requested minimum rate if the available bandwidth permits. An extra condition is added here: the delay bound constraints for all selected streams must not be violated based on Theorem 2. The two conditions are jointly applied to each stream until either the total available bandwidth is exhausted or all the streams are examined.

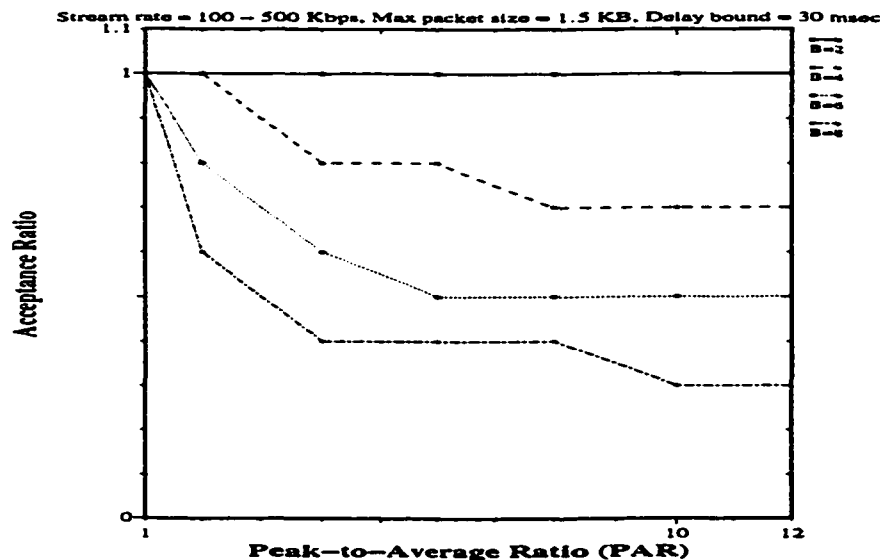


Fig. 3.11. Effect of peak-to-average ratio on acceptance ratio.

### 3.5.3 Simulation results

In this section, we present results from simulation experiments conducted to investigate the effect of using a traffic characterization model as M-LBAP, and enforcing delay constraints, on the performance of the range-based model for resource allocation. In each experiment, the session was composed of identical streams with average rate in the range from 100 to 500 Kbps. The values used for the other three parameters of the M-LBAP model are indicated on the charts. The number of streams requested to be activated was set to the maximum number that can be admitted based on the rate constraint alone, i.e.,  $\sum_{i=1}^n R_{min_i} = R_{tot}$  for the requested streams. In what follows the performance of RISA only is discussed as a representative of the range-based model, since the performance of I-WFS was always found to track closely that of RISA.

Figure 3.11 shows the effect of the *PAR* parameter on the acceptance ratio

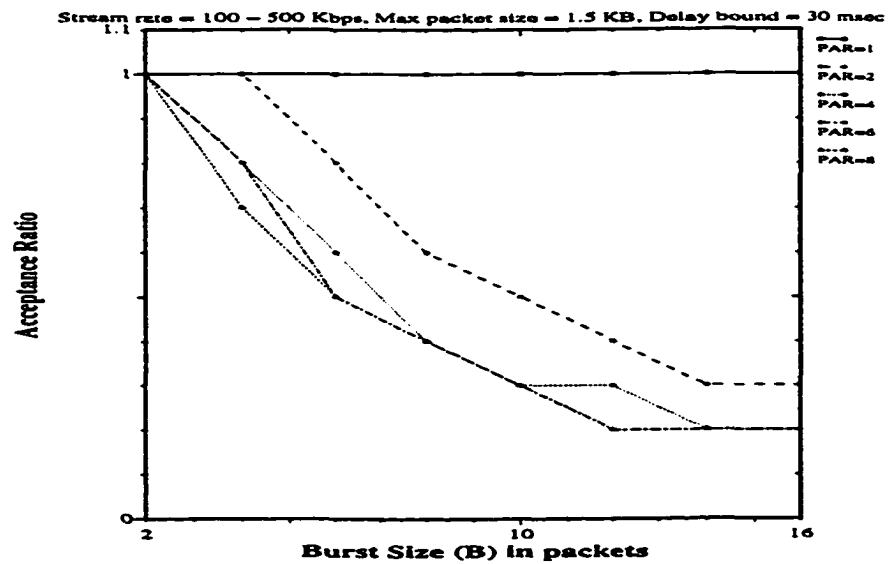


Fig. 3.12. Effect of burst size on acceptance ratio.

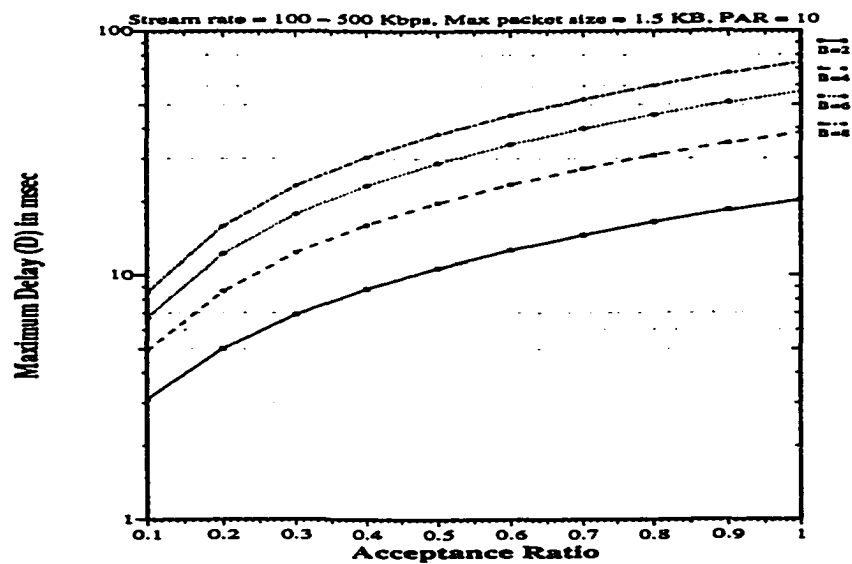


Fig. 3.13. Variation of maximum delay with acceptance ratio.

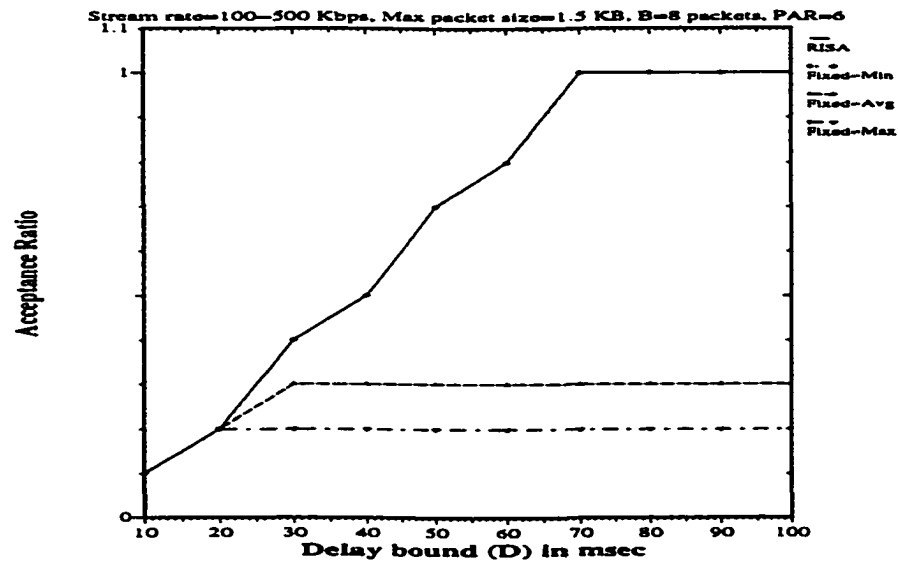


Fig. 3.14. Effect of delay bound on acceptance ratio.

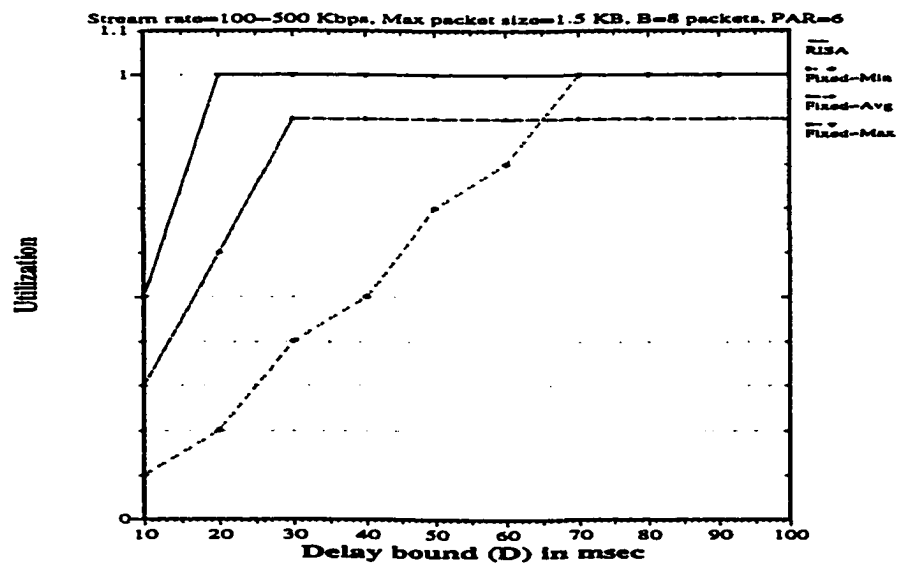


Fig. 3.15. Effect of delay bound on utilization.

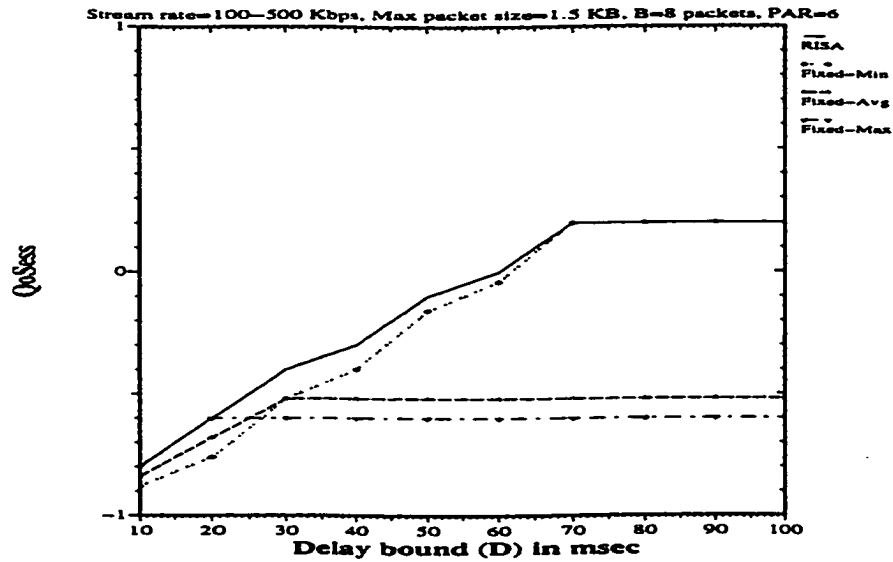


Fig. 3.16. Effect of delay bound on QoSess.

for the RISA approach. The acceptance ratio is defined as follows [35].

$$\text{Acceptance ratio} = \frac{\text{Number of accepted (activated) streams}}{\text{Number of streams requested to be activated}}$$

It is clear from the figure that the effect of the  $PAR$  parameter almost stabilizes for values above 5. This relaxes the requirement for exact calculation of  $PAR$ , which is an advantage for using  $PAR$  instead of the peak rate as the fourth parameter for the M-LBAP model. A rough estimate for  $PAR$  can be easily obtained by dividing the maximum frame size by the average frame size.

Figure 3.12 shows that the value of the burst size  $B$  strongly affects the acceptance ratio. Also, it emphasizes the fact that, for larger values of  $PAR$ , the acceptance ratio is less dependent on the accuracy of the  $PAR$  estimate.

In Figure 3.13, the maximum delay is computed as a function of the acceptance ratio, for different burst sizes. The figure indicates that higher acceptance ratios can be achieved at the same reserved total bandwidth for the group of streams by relaxing the delay bound.

In order to evaluate the benefits of employing degradation paths in inter-stream adaptation, the RISA approach was compared to the three fixed-point policies that were described before. Figures 3.14 and 3.15 show that while some of the fixed approaches achieve high utilization ratios and others achieve high acceptance ratios, RISA strikes the balance of achieving both goals. This is reflected on the QoSess metric, as shown in Figure 3.16. As mentioned before, the number of streams requested to be activated was set to be equal to the maximum number that could be admitted based on the rate constraint alone. This explains why the QoSess values for the *min* fixed-point policy are close to those for RISA. Typically, during a session there will be periods where the number of requested streams is smaller and hence significantly higher QoSess values will be obtained using RISA relative to the *min* fixed-point policy.

From the above results, we conclude that the RISA and I-WFS range-based policies maintain their superiority in managing IMC sessions over fixed-point allocations, in presence of delay bound constraints. Accommodating delay bounds requires tight traffic characterization. It was shown that the M-LBAP model provides a simple way for tight traffic characterization without imposing the need for extensive analysis for estimation of the traffic parameters.

### 3.6 Inter-Stream Adaptation in a Best-Effort Environment

Operating in a best-effort environment imposes an additional constraint on the participants of an IMC session: the available bandwidth/capacity of a receiver is not known or fixed beforehand by a reservation protocol. In addition, another constraint which should be accounted for is that multimedia sources can typically change their transmission rates in discrete steps only.

An inter-stream bandwidth adaptation algorithm which accounts for the fact

that sources can change their rates in discrete steps only, and which is intended for deployment over the current best-effort MBone, was presented by Amir et al. in [4]. The algorithm maps the layers of the streams to fictitious channels with fixed capacities. The channel packing effect is an obvious drawback in that approach, which may lead to inefficiencies in utilizing the available bandwidth. Another drawback to the concept of channels is that the receiver may have to join (or leave) multiple layers, assigned to the same channel, simultaneously in the adaptation process which may introduce strong fluctuations that may lead to instability. Moreover, the algorithm requires knowledge about the maximum session bandwidth. This is not a problem by itself as an upper bound can always be estimated. However, the allocation of layers to channels depends heavily on the session bandwidth input to the algorithm. This leads to unfair allocation of bandwidth among streams for the low end receivers, violating one of the most important declared objectives of the algorithm. This unfairness becomes more prominent as heterogeneity among receivers increases and the gap between the capacity of the low end receivers and the session maximum bandwidth increases.

Our objective in this section is to devise two algorithms, which approximate the behavior of I-WFS and RISA under the two additional constraints mentioned above, while avoiding the problems identified in Amir's algorithm. It can be easily shown that RISA, without any modifications, can support the above two constraints. However, devising an algorithm *A-IWFS*, which approximates I-WFS under these constraints, is more involved.

The first of the above two constraints implies that each receiver has to estimate its own capacity by progressively increasing its level of subscription to different streams and observing the effect of such subscription on performance, until it reaches a stable point. At any point a receiver stabilizes, the share of each source should be as close as possible to the share which would have been allocated by I-WFS if the available bandwidth at this point was known beforehand. This objective is even harder to achieve under the second constraint which poses an extra level of difficulty



TABLE 3.1  
NOTATION USED IN THE A-IWFS ALGORITHM

$S$	list of sources
$N$	number of sources
$L_{tot}$	total number of distinct layers from all sources
$B$	cumulative bandwidth allocated so far
$p_k$	priority of source $k$
$L_{k,l}$	layer $l$ of source $k$
$R_{k,l}$	rate of layer $l$ of source $k$
$L_{next_k}$	next layer to process from source $k$
$L_{max_k}$	highest layer of source $k$
$Order_{k,l}$	the position of layer $l$ of source $k$ in the linear order (starting from 1)
$Sched_k$	the earliest point (smallest value of $B$ ) at which the next layer from source $k$ can be processed (assigned an order)

in allocating the shares in accurate accordance to the priorities of the sources.

The A-IWFS algorithm produces a linear order of layers from all sources in the session. In order to do that, it uses knowledge about the priorities of the different streams together with knowledge about the discrete increments/decrements in operating points of each stream. Each receiver follows that linear order of layers. It cannot subscribe to a layer of higher order unless it has already subscribed to all lower order layers, and vice versa. Table 3.1 summarizes the notation used in the A-IWFS algorithm. The algorithm itself is listed in Figure 3.17, and its complexity and correctness are given in Appendix A.

### 3.6.1 Results and evaluation

The objective of this section is to evaluate the effectiveness of the A-IWFS algorithm in allocating the bandwidth available to a receiver among all the streams of the session, in a fair way. In addition, we compare our two algorithms, A-IWFS and

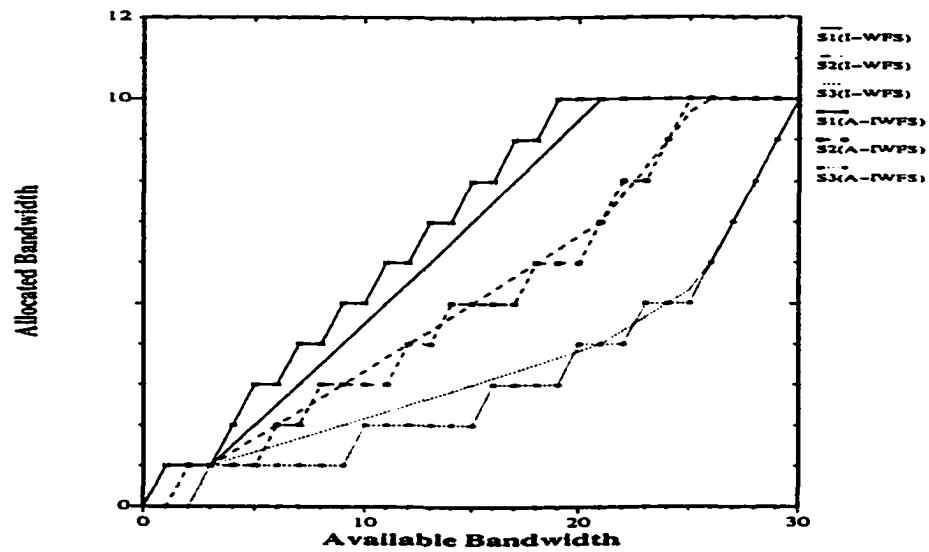
```

A-IWFS() {
    sort the list of sources  $S$  in descending order of  $p_i$  ;
     $tot\_p = \sum_{i=1}^N p_i$  ;  $ord = 1$  ;  $B = 0$  ;
    while (  $ord \leq Ltot$  ) {
        if (  $ord \leq N$  ) {
             $k = S_{ord}$  ;// get the next source in the sorted list
             $Lnext_k = 1$  ;
             $Sched_k = B$  ;
        } else {
             $k = Arg\ Min_{\forall i: p_i \neq 0} \{ Sched_i \}$  ;
             $Lnext_k = Lnext_k + +$  ;
        }
         $l = Lnext_k$  ;
         $Order_{k,l} = ord + +$  ;
         $Sched_k = Sched_k + \frac{tot\_p}{p_k} R_{k,l}$  ;
         $B = B + R_{k,l}$  ;
        if (  $Lmax_k == l$  ) {
             $tot\_p = tot\_p - p_k$  ;
            Recompute_All_Schedules( $k$ ) ;
        }
    }
}

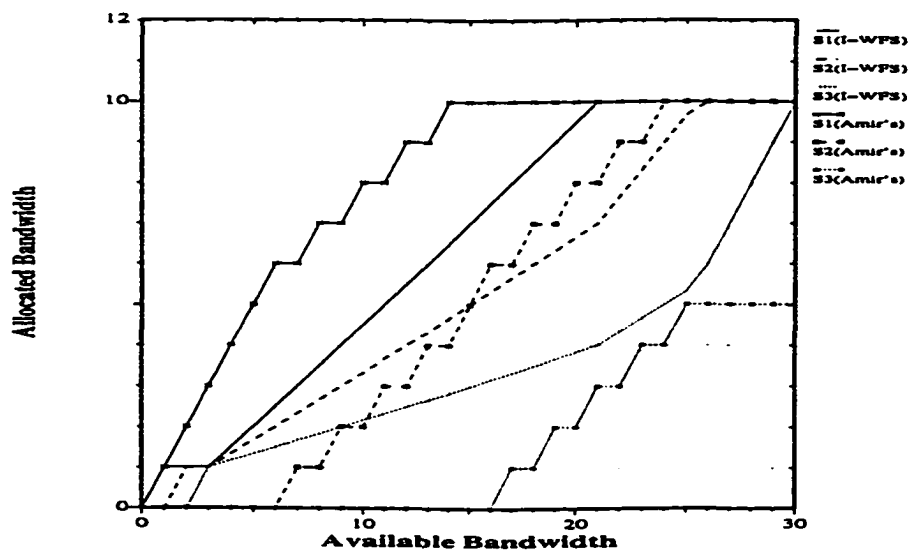
Recompute_All_Schedules( $k$ ) {
     $p = p_k$  ;  $p_k = 0$  ;
    for-each source  $i$  s.t.  $p_i \neq 0$  {
         $l = Lnext_i$  ;
         $Sched_i = Sched_i - \frac{p}{p_i} R_{i,l}$  ;
    }
}

```

Fig. 3.17. The A-IWFS algorithm for resource allocation.



(a)



(b)

Fig. 3.18. Comparing bandwidth allocation to I-WFS. (a) A-IWFS. (b) Amir's algorithm.

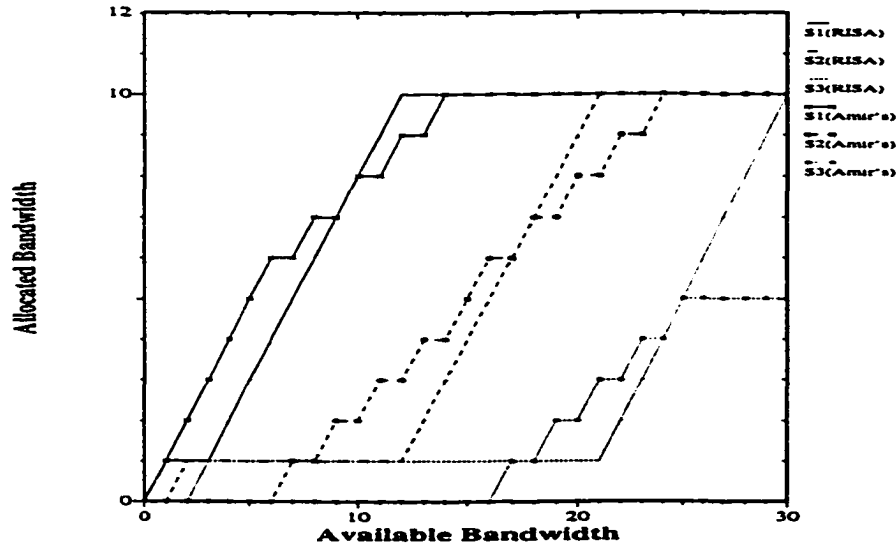


Fig. 3.19. Comparing bandwidth allocation by Amir's algorithm to RISA.

RISA, to Amir's algorithm [4]. In this comparison, we used channels of capacity equal to one unit of bandwidth in order to avoid penalizing Amir's algorithm by the channel packing effect. The total number of channels was chosen such that all layers can be accommodated by the high-end receivers in the session. We simulated an IMC session composed of three streams S1, S2, and S3. Their weights were set to 0.5, 0.333, and 0.167, respectively, with S1 being the most important. Each stream has a maximum of 10 layers each requiring 1 unit of bandwidth.

Since A-IWFS is intended to approximate the performance of I-WFS, under the two constraints specified in the previous section, we compare the bandwidth allocation devised by A-IWFS to that devised by I-WFS. In order to do this comparison, we first run A-IWFS and obtain its output which is a linear order of all the layers from all the sources. Then, we repeatedly run I-WFS for each comparison point. In each run of I-WFS, we set a certain value for  $B$ , the available bandwidth, and observe the I-WFS allocation of  $B$  among the streams. The corresponding allocation

done by A-IWFS can be obtained by truncating the linear order before the layer by which the cumulative bandwidth exceeds  $B$ , and observing the share of each stream in that portion of the linear order. Figure 3.18(a) depicts the bandwidth shares of the 3 streams as allocated by A-IWFS and by I-WFS. It is clear from the figure that A-IWFS tracks well the I-WFS allocation, in spite of its operation under more constraints.

Figure 3.18(b) depicts the bandwidth share for each of the above 3 streams as obtained by Amir's algorithm, in contrast to the I-WFS case. As can be seen from the figure, Amir's allocation is far from that of I-WFS, i.e., the session bandwidth is not shared fairly among the streams. The deviation from the I-WFS allocation exceeds 30% in some cases.

Figure 3.19 compares the allocation of Amir's algorithm to RISA. Although Amir's allocation is closer to RISA than to I-WFS (deviation does not exceed 20% before saturation), yet it generally has two major drawbacks relative to our two algorithms. First, for the high-end receivers, some of the streams may saturate in spite of the availability of bandwidth leading to under utilization of resources, as is the case with S3 in this experiment, which leaves over 16% of the available bandwidth non-utilized. Second, for the low-end or congested receivers, the number of active streams may be low and some streams may not be granted their initial base layer until after other streams are well enhanced, e.g., in this experiment, a receiver with 6 units of available bandwidth will not receive any layers from S2 or S3, and all the available bandwidth will be dedicated to S1.

From the above, we conclude that A-IWFS achieves better utilization of bandwidth, more fairness in allocating the bandwidth, and maximizes the number of admitted streams. It is more suitable than the other algorithms for best-effort networks, as it does not require reservation or prior knowledge about the available bandwidth.

### 3.7 Conclusions

In this chapter, we focused on one important component of the QoSess control layer; the inter-stream bandwidth adaptation mechanism. Quality of session control is primarily achieved by means of an inter-stream adaptation mechanism that accommodates application semantics, and is driven by the instantaneous relative importance of the different streams to the session. A QoSess graph is used to represent the relative importance of the different streams to the session. The QoSess graph enables the separation of inter-stream adaptation policies from mechanisms.

In order to show the advantages of inter-stream adaptation, we abstracted the problem as a simplified resource allocation problem. We compared two generic resource allocation models. In the first model, clients request a certain fixed level of the resource from the resource manager. This is the commonly used approach in admitting connections in networks providing QoS support on individual connections basis. In the second model, clients can operate at any point of a range of possible resource allocations. This model matches the cooperative nature assumed among the streams of an application.

We proposed two policies, RISA and I-WFS, for approaching the range-based resource allocation problem, and introduced a unified metric, *QoSess*, for comparing the effectiveness of resource allocation strategies, in terms of the aggregate level of satisfaction of all the clients.

The behavior of the two models was contrasted using two types of traffic: constant bit rate traffic; and traffic characterized using the M-LBAP model. The simulation study that was conducted confirmed that the range-based model is more suitable for groups of streams which are cooperating to fulfill a unified global goal, and each is willing to sacrifice for the sake of the benefit of the whole group. It was shown that better resource utilization and acceptance ratios are always achievable using RISA and I-WFS relative to fixed-point allocations. These achievable results were reflected on and summarized by the introduced QoSess metric.

In the absence of group reservation support in the network, the inter-stream bandwidth adaptation mechanism used should be able to operate correctly without knowledge about the bandwidth available to the session. Additionally, multimedia sources are typically able to vary their transmission rates in discrete steps only. The RISA algorithm was shown to support these two constraints, and a new inter-stream bandwidth adaptation algorithm, A-IWFS, was devised to approximate the behavior of I-WFS under these additional constraints. The performance of the new algorithm was studied, and its efficiency and fairness in utilizing the bandwidth available to a session were demonstrated by simulation results.

While the next chapter focuses on the feedback component of the quality of session framework, Chapter 5 realizes this framework by means of an architecture which incorporates the presented inter-stream adaptation techniques.

:

## CHAPTER IV

### STATE FEEDBACK PROTOCOL

In this chapter, we present one of the main building blocks of the QoSess control layer; a state feedback protocol. This protocol provides the source of a multimedia stream with deterministic information regarding the state of the receivers. The state of a receiver may be defined as the layers which it is interested in receiving from the source of a hierarchically encoded stream. Given this knowledge, the sender can suppress or start sending the correct layers. The feedback mechanism is not only important for saving the source host and LAN resources but for saving WAN resources as well in situations where the addressing scheme used for the layers of the IMC application does not permit the intermediate routers to suppress unwanted layers, or where the IMC session is conducted over an Intranet whose subnets are inter-connected via low level switches that do not implement the IGMP protocol [24] for suppressing multicast packets for which no receivers exist on the subnet, which is not an uncommon setup for IMC applications (see for example [45]). Soliciting feedback from receivers in a multicast group might create a *reply implosion* problem, in which a potentially large number of receivers send almost simultaneous redundant replies. We present a scalable and robust solution to this problem.

The rest of this chapter is organized as follows. In Section 4.1, the role of feedback in different adaptive multimedia multicast systems is illustrated. The proposed feedback protocol is described in detail in Section 4.2, followed by a performance study and comparison in Section 4.3. In Section 4.4, adaptive enhancements



for the proposed protocol in order to support very large groups of receivers are described, and we present our conclusions in Section 4.5.

## 4.1 Feedback Role in Adaptive Multimedia Multicast Systems

Early attempts towards providing adaptive transport of multimedia streams over the Internet focused on the sender as the entity playing the major role in the adaptation process [9, 10, 12]. Information about the congestion state of the network, as seen by the receivers, was fed-back to the sender which used it to adapt to changes in the network state. In many cases, the monitored performance parameters (e.g., loss rate, delay, jitter, throughput) were mapped, by the receiver, to one of several qualitative performance levels, and reported to the sender [9, 12, 16]. The sender adapted its transmission rate by varying the quality of the transmitted media content by means of controlling several encoder parameters (e.g., frame rate, frame size, or quantization step for video streams). The sender often based its decisions on the worst case state reported [12], and sometimes based it on a threshold of the number of receivers suffering the worst state [9]. In this approach all receivers have to receive the same quality of multimedia streams regardless of the differences in their capabilities and the capacities of the network connections leading to them. Although sometimes it is desired to maintain identical stream quality across all participants of a session (e.g., for some discrete media streams), yet this is not always the case especially with continuous media streams.

The first approach, to address the need for providing a multi-grade service to participants of the same session, was represented by the introduction of the concept of *simulcast* [42, 59]. In a simulcast system, the sender simultaneously multicasts several parallel streams corresponding to the same source, but each is encoded at a different quality level. Each receiver joins the multicast group that matches its

capabilities. Within a group, the same techniques of source adaptation, that were mentioned above, are applied within a limited range. Thus, the same feedback mechanisms are also deployed within each group.

With the advent of hierarchical encoding techniques [47, 52], a new trend in adaptive multimedia transport appeared in which the receiver plays the sole role in adaptation [46]. In such systems the receiver is responsible for determining its own capabilities, and consequently, it selects the number of layers to receive from the hierarchically encoded stream. The source, however, is assumed to be constantly multicasting all the layers.

While it is very obvious that the layered encoding approach is more efficient in the utilization of resources relative to the simulcast approach, yet it is still debatable whether layered encoding techniques will be able to provide the same media quality as the simulcast encoders which operate in parallel, each optimized for a particular target rate. In spite of this debate, the layered approach is the most appealing from the networking point of view, due to its efficient utilization of network resources, especially bandwidth. However, this approach as described is not as efficient as can be. The fact that the source keeps sending at full rate, all layers, constantly, may lead to the waste of resources, in the case where no receivers subscribe to some of the layers. On the other hand, augmenting this approach with a simple scalable feedback mechanism that provides the source with information regarding which layers are being consumed and which are not, yields more efficiency in resource consumption, as the sender can get actively involved in the adaptation process by suppressing the unused layers.

The introduction of such a feedback mechanism, for receiver-oriented layered transport of multimedia streams, is not only an added efficiency feature for such transport protocols, but it is also a critical feature for the success of IMC sessions in which multiple streams are concurrently active. In such collaboration sessions, multiple streams are typically distributed to all participants of the session, and the overall session quality is determined by the quality of each of the streams

as well as by their relative importance and contribution to the on-going activity. In presence of scarce resources, it is logical to sacrifice the quality of one low priority stream for the sake of releasing resources to be used by a higher priority stream, as explained in Chapter 3. Should the low priority source keep pushing all unused layers to the network, the decision made by the receivers to drop these layers for releasing resources is rendered useless. This uselessness will hold true forever for the source host and LAN, while the rest of the network may eventually have these resources released as the multicast routers stop forwarding the unused layers. In situations where the application's addressing scheme for the layers does not permit the intermediate routers to suppress unwanted layers, WAN resources may also be wasted. Besides the unnecessary delay in releasing resources, the fact that the source host and LAN will always be overloaded is very critical, as the session participants on this LAN may not be able to receive other higher priority streams. The problem is more crucial for Intranet-based collaboration systems since all the session participants (senders and receivers) are typically within a few hops from one another [2, 45]. In addition, it is not uncommon to conduct such sessions over an Intranet that does not contain routing elements that are capable of suppressing unwanted traffic by deploying the IGMP protocol [24].

Moreover, since the sender may be sending only a subset of its layers, it needs to know about the existence of clients for higher layers that are currently suppressed, as soon as these clients subscribe to these layers. This information must be provided to the sender in a timely and scalable way that avoids potential implosion problems in such cases when many clients subscribe to higher layers almost simultaneously. This is likely to happen when some streams are shutdown releasing resources that can be utilized by other active streams.

From the above we conclude that a feedback mechanism is necessary for involving the sender in the adaptation process for receiver-driven layered multicast of multimedia streams, especially in the context of collaborative multimedia sessions. Moreover, such a feedback mechanism is essentially the same as, and can replace,

feedback mechanisms for supporting simulcast and single-rate multicasts. In the following section, we introduce our proposed robust mechanism for providing scalable feedback in adaptive multimedia multicast systems.

## 4.2 A Scalable Feedback Mechanism

In this section, we describe the proposed mechanism for eliciting feedback information from the receivers in a multicast group. The objective of the algorithm is to find out the worst case state among a group of receivers. The definition of the worst case state is dependent upon the context in which the feedback mechanism is applied. It can be the network congestion state as seen by the receivers. This may be useful for applications where a similar consistent view is required for all the receivers, and the source is not capable of providing a multi-grade service, and hence must adapt to the receiver experiencing the worst performance. Another definition, of worst case state as seen by all receivers, is identifying the highest layer a receiver is expecting to receive in a hierarchically encoded stream. This allows the sender to adjust its transmission rate in order not to waste resources on layers that no receiver is subscribing to, and to start sending previously suppressed layers as soon as receivers subscribe to receive them. This is particularly important in the context of managing multimedia streams in collaborative sessions, because in such sessions the sender of a stream is typically simultaneously receiving multiple streams, and hence the assumption that the sender has abundant resources is not valid.

In the rest of the chapter, we assume that at every instant in time each receiver is in one state  $s$ , where  $s = 1, 2, \dots, H$ .  $H$  is the highest or worst case state, and the state of a receiver may change over time.

We consider the general case when neither the group size nor the round-trip time from the sender to each receiver is known. As will be shown later, this information is not necessary as the mechanism estimates the average round trip time in the group, and uses it to adjust its timeout periods.

In the proposed mechanism, the sender sends one type of probe messages, called *SolicitReply* messages, on a special multicast group which the sender and all the receivers join. The probe message contains a *RTT* field, which contains an estimate for the average round trip time from the sender to the group members. Upon receiving the *SolicitReply* probe, a receiver sets a timer to expire after a random delay period which is drawn from the interval

$$\left[ C_1 f(s) \frac{RTT}{2}, (C_1 f(s) + C_2 g(s)) \frac{RTT}{2} \right],$$

where  $f(s)$  and  $g(s)$  are two non-increasing functions of the state  $s$ ,  $C_1$  and  $C_2$  are two parameters whose values are discussed later in detail. The receiver then keeps listening to the multicast group. If the timer expires, the receiver multicasts a reply message to the whole group. The reply message contains the state information as seen by this receiver (e.g., highest layer expected to receive in a hierarchically encoded stream). On the other hand, if the receiver receives another receiver's reply before its timer expires and that reply contains either the same or higher (worse) state, then the receiver cancels its timer and suppresses its own reply. This implies the need for careful selection of  $f(s)$ ,  $g(s)$ ,  $C_1$ , and  $C_2$  in order to avoid the reply implosion problem, while maintaining a low response time. In the subsequent subsections, we discuss in detail choices for  $f(s)$ ,  $g(s)$ ,  $C_1$ ,  $C_2$ , and *RTT*.

#### 4.2.1 Selecting the timeout functions

The objective of setting the timeout periods as a function of  $f(s)$ , and  $g(s)$  is to distribute the timeouts as in Figure 4.1. Receivers in higher states randomize their timeouts over periods that start earlier than receivers in lower states, thus allowing for higher state responses to suppress lower state responses. In addition, the lower state receivers randomize their timeouts over longer periods relative to higher state receivers. This is because as time elapses and no responses are generated this means that the distribution of receivers over states is biased and more receivers belong to

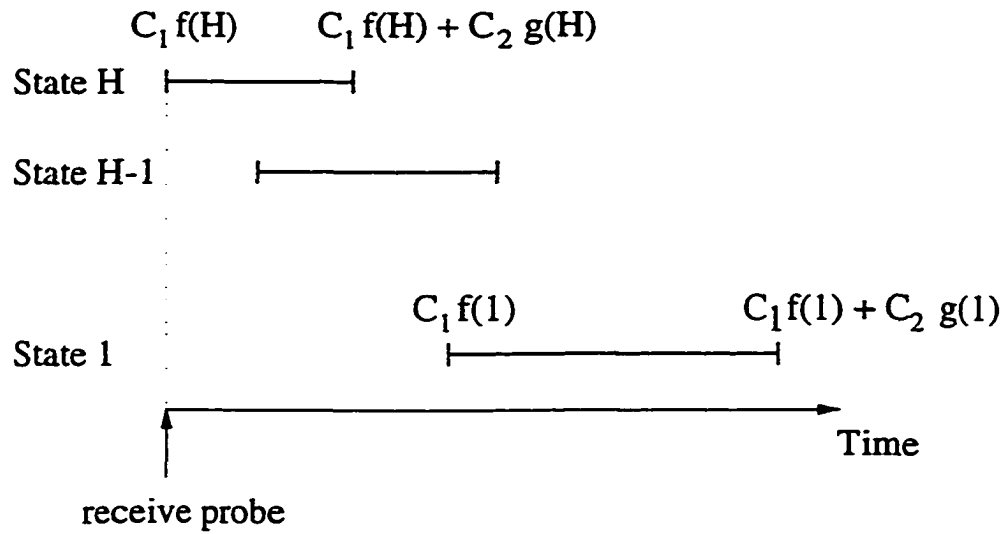


Fig. 4.1. Distribution of timeout periods according to receiver state.

the lower states. Thus it is desired to randomize these condensed replies over longer periods.

In order to meet these objectives,  $f(s)$  and  $g(s)$  must be non-increasing functions of  $s$ . Also,  $f(H)$  should equal 0 to avoid unnecessary delays in response time, while  $g(s) > 0$  must be satisfied for all values of  $s$  to allow for randomization of timeout periods. We chose to make  $f(s)$  and  $g(s)$  linear functions in  $s$  in order to avoid excessive delays in response time, where  $f(s) = H - s$ , and  $g(s) = f(s) + k = H - s + k$ .

The parameters  $C_1$  and  $C_2$  scale the functions  $f(s)$  and  $g(s)$ .  $C_1$  controls the aggressiveness of the algorithm in eliminating replies from lower state receivers, while  $C_2$  controls the level of suppression of redundant replies from receivers in the same state. The values of these two parameters are explored in depth in the following sections. The value of  $k$  is set to 1. Selecting the value of  $k$  is not critical, since the parameter  $C_2$  scales  $g(s)$ , and the value of  $C_2$  can be tuned to optimize the performance of the mechanism given the selected value of  $k$ .

### 4.2.2 Exploring the parameter space

In this section, we attempt to find bounds for the ranges of operation of the parameters  $C_1$  and  $C_2$ . Obviously, low values for  $C_1$  and  $C_2$  are desired in order to reduce the response time. On the other hand, excessive reduction in the value of either of the two parameters may lead to inefficiency in terms of the number of produced replies possibly leading to a state of reply implosion.

In order to effect a shift in the start time of the timeout periods based on the state of the receiver, as in Figure 4.1,  $C_1 > 0$  must be satisfied for all  $s < H$ . This shift allows for the high state replies to suppress low state replies. Similarly,  $C_2 > 0$  must be satisfied for all values of  $s$ , in order to allow for randomization of timeout periods for receivers belonging to the same state, thus enabling suppression of redundant replies which carry the same state information.

To further bound the values of  $C_1$  and  $C_2$ , we analyze two extreme network topologies, namely: the chain and the star topologies. Given a certain distribution of receiver distances from the sender, the feedback mechanism exhibits worst case performance, in terms of the number of redundant replies, when the receivers are connected in a star topology with the sender at its center. This is because connecting those receivers in a star topology maximizes the distance between any pair of receivers, to the sum of their distances from the sender, and hence minimizes the likelihood of suppression of redundant replies. On the contrary connecting those receivers in a chain topology minimizes the distance between any pair, to the difference between their distances from the sender, and hence maximizes the likelihood of suppression of redundant replies. Therefore, for a given distribution of distances, and an arbitrary topology, the performance of the feedback mechanism lies somewhere in between the chain and the star cases.

Figure 4.2 further illustrates this issue. Given  $r_1$  and  $r_2$  which are the distances between the sender and each of the two receivers,  $R_1$  and  $R_2$ , respectively. It can be easily shown that the star topology maximizes the distance,  $d$ , between

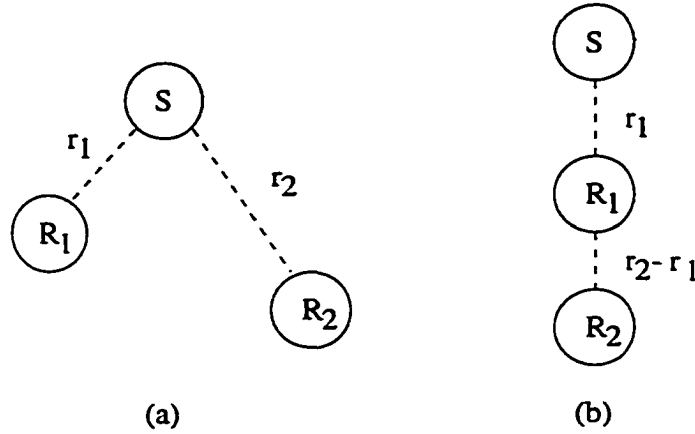


Fig. 4.2. The extreme topologies. (a) Star. (b) Chain.

$R_1$  and  $R_2$  to the sum  $r_1 + r_2$ , assuming symmetric bidirectional delays and shortest path routing. In order to prove that the chain topology minimizes the distance between  $R_1$  and  $R_2$ , let an independent path with delay  $d$  smaller than  $r_2 - r_1$  exist between  $R_1$  and  $R_2$ . This contradicts with shortest path routing, since  $d + r_1$ , rather than  $r_2$ , constitute the shortest path from  $R_2$  to the sender in this case. Therefore, the chain topology minimizes the distance  $d$  between  $R_1$  and  $R_2$  to be equal to the difference between their respective distances from the sender.

### Chain topology

In the chain topology, the sender is at one end of a linear list of nodes. The rest of the nodes in the list are receivers. Let  $r = \frac{RTT}{2}$  be a bound on the one way distance from the sender to any of the receivers or vice versa. Let the sender send a probe at time  $t$ . The farthest receiver receives the probe at time  $t + r$ . If this receiver is the only one in the highest state, and if it emits its reply as soon as it receives the probe, then all other receivers will have heard this reply by time  $t + 2r$ . In order to suppress all replies from lower state receivers in this case,  $C_1 \geq 2$  must be satisfied.  $C_1 = 2$  makes the difference between the start time of two successive states equal to  $2r$ .



### Star topology

In the star topology, the sender is connected to each receiver by a separate link. Any message sent from one receiver to another passes through the sender's node. Let all the receivers be at a distance  $r = \frac{RTT}{2}$  from the sender. Thus the distance between any two receivers is equal to  $2r$ .

Let  $G_s$  be the number of receivers in state  $s$ , and let  $T_s$  be the first timer to expire for receivers in state  $s$ . The expected value of  $T_s$  is  $(C_1 f(s) + \frac{C_2 g(s)}{G_s})r$ , since  $G_s$  timers are uniformly distributed over a period of  $C_2 g(s)r$ .

For receivers having the same state, if the first timer expires at time  $t$ , then all the timers that are set to expire in the period from  $t$  to  $t + 2r$  will not be suppressed, and all those that are set to expire after  $t + 2r$  will be suppressed. Therefore, the expected number of timers to expire is equal to 1 plus the expected number of timers to expire in a period of length  $2r$ , which is equal to  $1 + \frac{2G_s}{C_2 g(s)}$ . Looking at the case of  $s = H$ , since  $g(H) = 1$ , then setting  $C_2$  to any value less than 2 does not allow for suppression of any of the redundant replies from receivers in state  $H$ . Thus  $C_2 > 2$  must be satisfied. In order to suppress all replies from receivers in state  $s - 1$ , we must have:

$$\begin{aligned}
 T_s + 2r &\leq T_{s-1} \\
 \text{or } (C_1 f(s) + \frac{C_2 g(s)}{G_s})r + 2r &\leq (C_1 f(s-1) + \frac{C_2 g(s-1)}{G_{s-1}})r \\
 \text{or } \frac{g(s)}{G_s} - \frac{g(s-1)}{G_{s-1}} &\leq \frac{C_1 - 2}{C_2} .
 \end{aligned}$$

For values of  $G_s$  and  $G_{s-1}$  which are relatively larger than  $g(s)$  and  $g(s-1)$ , we get  $C_1 \geq 2$ , which is the same condition for  $C_1$  which we obtained from the chain topology. In Section 4.3, we explore the effect of  $C_2$  on the performance of the feedback mechanism using simulation experiments.

### 4.2.3 Estimating the round-trip time

To compute the average round-trip time from the sender to the group of receivers, every probe sent is time-stamped by the sender. That time-stamp is reflected in the reply message together with the actual delay period that the receiver waited before replying. This allows the sender to compute the round-trip time to this receiver. The smoothed average round-trip time,  $srtt$ , and the smoothed mean sample deviation  $rttvar$  are computed from the received round-trip time samples, using the same technique applied in TCP [40], as follows:

$$\begin{aligned} srtt &= \alpha srtt + (1 - \alpha) sample, & \alpha &= 7/8, \\ rttvar &= \beta rttvar + (1 - \beta) |srtt - sample|, & \beta &= 3/4. \end{aligned}$$

In TCP, the amount  $srtt + 4 rttvar$  is used in setting the retransmission timeouts in place of twice the round-trip time. As will be shown in Section 4.3, this amount is conservative and over estimates the average round-trip time to the group members. Instead we use only  $srtt$  as the estimate for average round-trip time. The recent value of  $srtt$  is carried in the  $RTT$  field of the next probe.

## 4.3 Simulation Study and Performance Comparison

In this section, we examine various issues, related to the performance and tuning of the feedback mechanism, using simulation. First we show the ability of the new feedback mechanism to eliminate the reply implosion problem as we explore the effect of  $C_2$  on its performance. Then we examine the accuracy of the round-trip time estimation algorithm. Finally, we further illustrate the scalability and robustness of the proposed feedback mechanism by contrasting it to an alternative candidate mechanism for feedback.

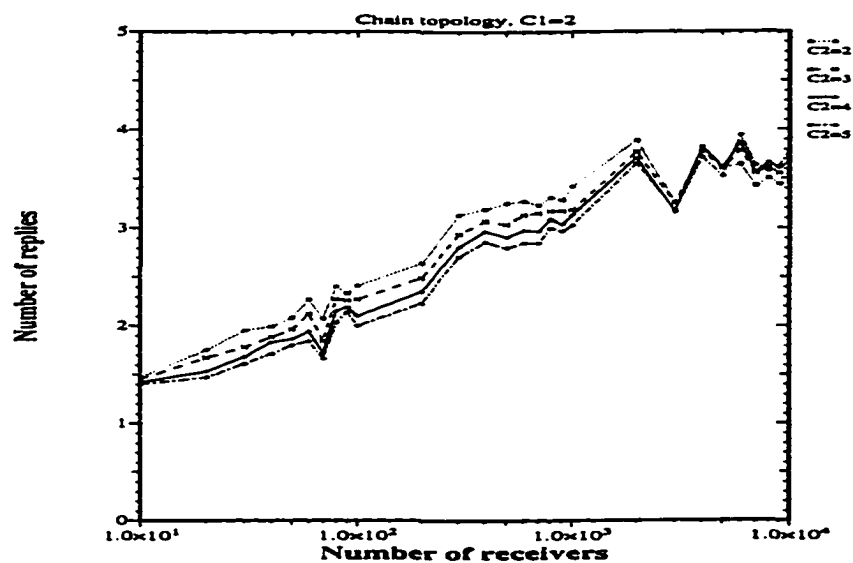
In order to address these issues, we ran several simulation experiments. Each experiment was setup as follows. The group size,  $G$ , and the maximum round-trip

time,  $RTT_{max}$ , were selected. Round-trip times uniformly distributed in the interval  $[0, RTT_{max}]$  were assigned to all the receivers, except the worst case state receivers whose round-trip times were uniformly distributed in the interval  $[t.RTT_{max}, RTT_{max}]$ , for investigating the effect of  $t$  over the performance, where  $0 \leq t \leq 1$ . The number of states,  $H$ , was set to 5, and each receiver was randomly assigned one of these states. The choice of 5 states (or layers) is reasonable as the state of the art hierarchical video encoders typically provide a number of layers in this range [46, 52]. Also, in applications where feedback information represents the perceived quality of service, typically 3 to 5 grades of quality are used [9, 12]. The feedback mechanism was simulated under the two extreme network topologies; the chain and the star.

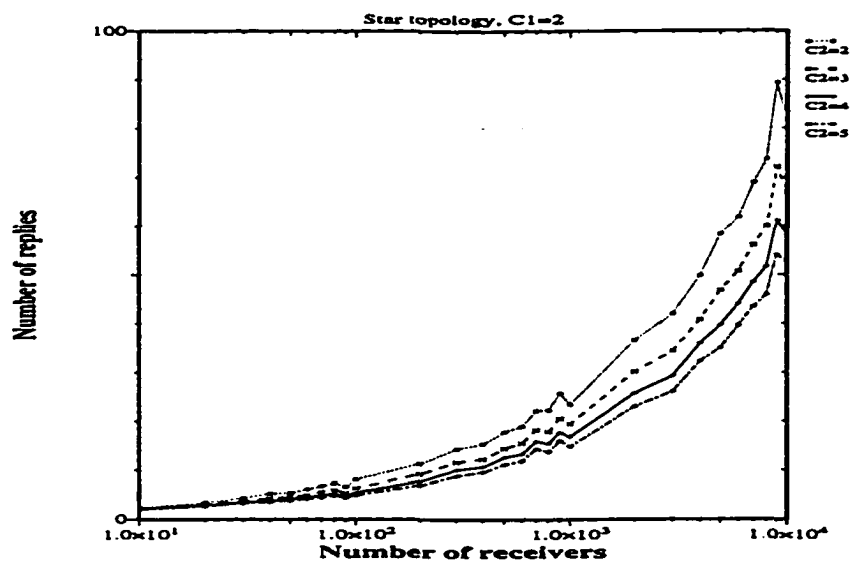
#### 4.3.1 Bounding constants in timing function

From the analysis in Section 4.2.2, we obtained the two conditions  $C_1 \geq 2$  and  $C_2 > 2$ . Setting  $C_1$  to its minimum value 2 eliminates replies from lower states, while avoiding unnecessary delays in response time. However, selecting an appropriate value for  $C_2$  is not as easy as such.

In Figure 4.3, the average number of replies is plotted for different values of  $C_2$ . The value of  $C_1$  was set to 2, for all the experiments in this section, and the average round-trip time was used in the  $RTT$  field of the probe messages. It is clear from the figure that the performance of the feedback mechanism is not sensitive to the value of  $C_2$  in the case of the chain topology. Also, the figure shows that the reply implosion problem is totally eliminated. Moreover, over 95% of the redundant replies were correct replies (i.e., worst case state replies) which shows the robustness of the mechanism in facing network losses and its efficiency in eliminating non-worst case replies. This also means that, practically, the sender may safely react according to the first received reply. Figure 4.4 depicts the corresponding average response times. The response time is measured at the sender, and represents the time from sending a probe until receiving the first correct reply. The response time behavior



(a)



(b)

Fig. 4.3. The effect of  $C_2$  on the number of replies. (a) Chain topology. (b) Star topology.

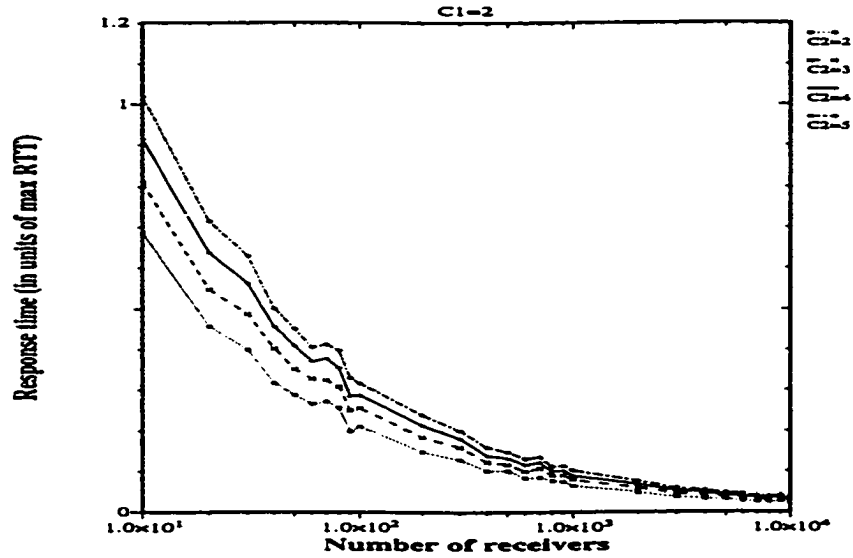
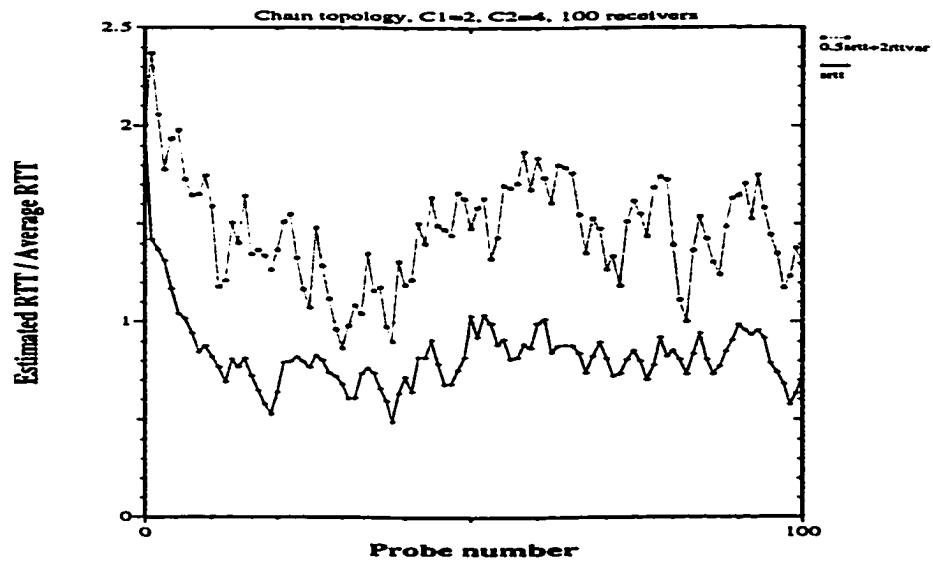


Fig. 4.4. The effect of  $C_2$  on response time.

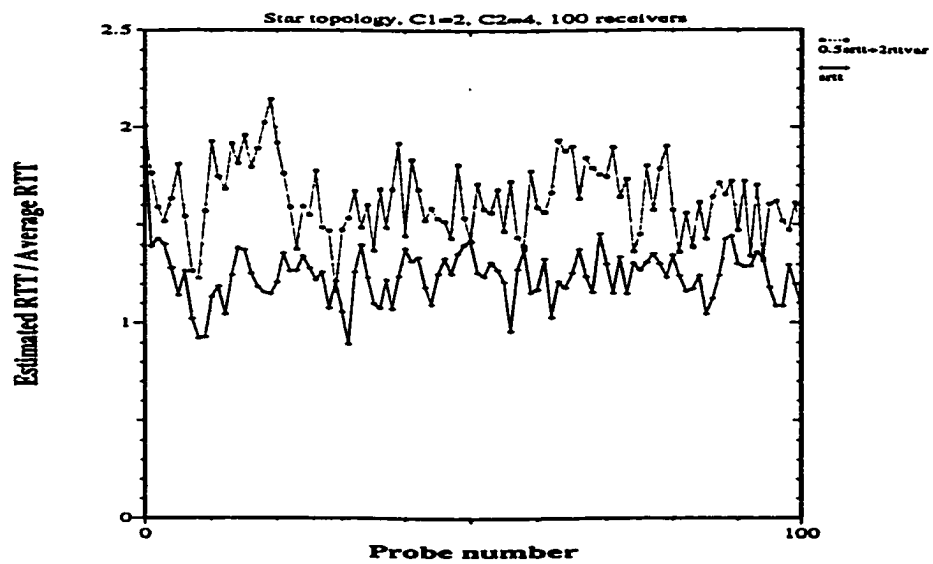
is the same for both topologies because it is dependent on the round-trip times distribution rather than on the topology. As shown in the figure, it is bounded from above by the maximum round-trip time to the group members.

These figures suggest that  $C_2 = 4$  is a reasonable setup.  $C_2 > 4$  does not significantly reduce the number of replies, while the response time increases. As can be seen from the figures, for typical sessions with up to 100 participants (e.g., IRI sessions [45]), less than 10% of the receivers reply to a probe, in the worst case, while for larger sessions of thousands of participants the reply ratio is below 1.5%.

It should be noted that the relative error in any of the presented average values does not exceed  $\pm 10\%$  with 95% confidence. This is true for all averages presented in this chapter.



(a)



(b)

Fig. 4.5. Accuracy of RTT estimate. (a) Chain topology. (b) Star topology.

### 4.3.2 Evaluating the round-trip time estimation technique

As mentioned in Section 4.2.3, the amount  $srtt + 4 rttvar$  is used in setting the retransmission timeouts in place of twice the round-trip time, in TCP. Figures 4.5(a) and (b) compare this approach to using only  $srtt$  as the estimate for average round-trip time. We chose to avoid the conservative approach of TCP, and to use only  $srtt$ , to avoid unnecessary prolonging of delay periods thus avoiding excessive delays in response time.

### 4.3.3 Performance comparison

Here, we further illustrate the scalability and robustness of the proposed feedback mechanism by contrasting it to an alternative candidate mechanism for feedback. The alternative mechanism uses the same approach taken by SRM [34] for discriminating between receivers in setting their timeout periods based on their individual distances from the source (i.e., timeouts are selected from the interval  $[C_1 d_i, (C_1 + C_2) d_i]$  where  $d_i$  is the one way distance from receiver  $i$  to the source). This, in turn, depends on the existence of session level messages for the distance estimation process as explained in Section 2.2.

Figure 4.6 and Figure 4.7 contrast the performance of our proposed feedback mechanism,  $A_1$ , to the alternative feedback mechanism,  $A_2$ , in the case of the chain topology. The graphs in Figure 4.6 depict the performance results when the worst case state receivers were distributed at distances in the range  $[0, RTT_{max}]$ , i.e.,  $t = 0$ , while the graphs in Figure 4.7 show the performance of the two algorithms when the worst case state receivers were distributed at distances in the range  $[0.2RTT_{max}, RTT_{max}]$ , i.e.,  $t = 0.2$ .

The figures show that the total messages sent in response to a probe in the case of the new feedback mechanism,  $A_1$ , is much lower than the total response plus session messages for the alternative feedback mechanism,  $A_2$ . As discussed in Section 2.2, the session overhead for  $A_2$  is dependent on the session bandwidth; we

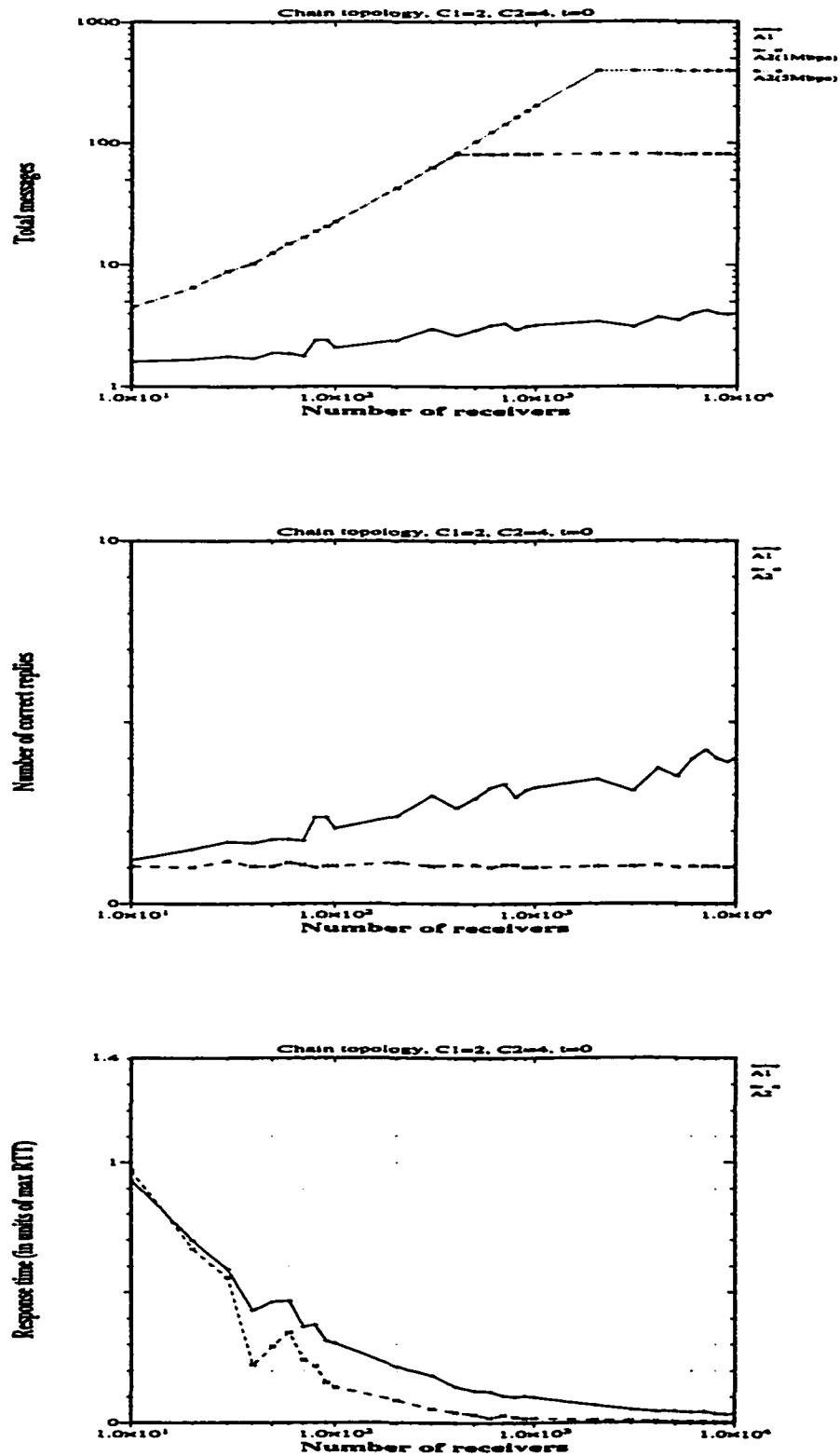


Fig. 4.6. Performance of the chain topology for  $t=0$ .



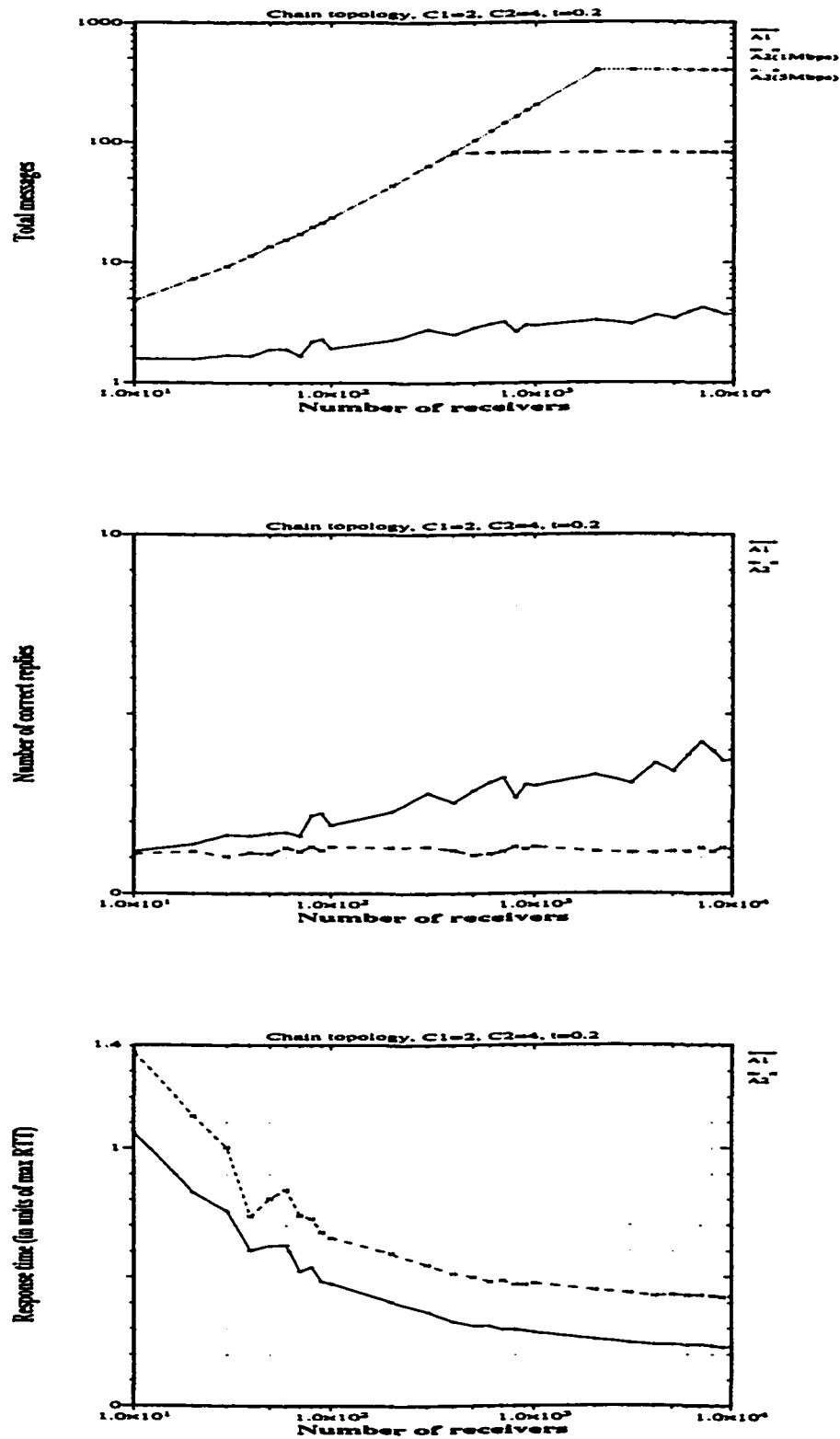


Fig. 4.7. Performance of the chain topology for  $t=0.2$ .

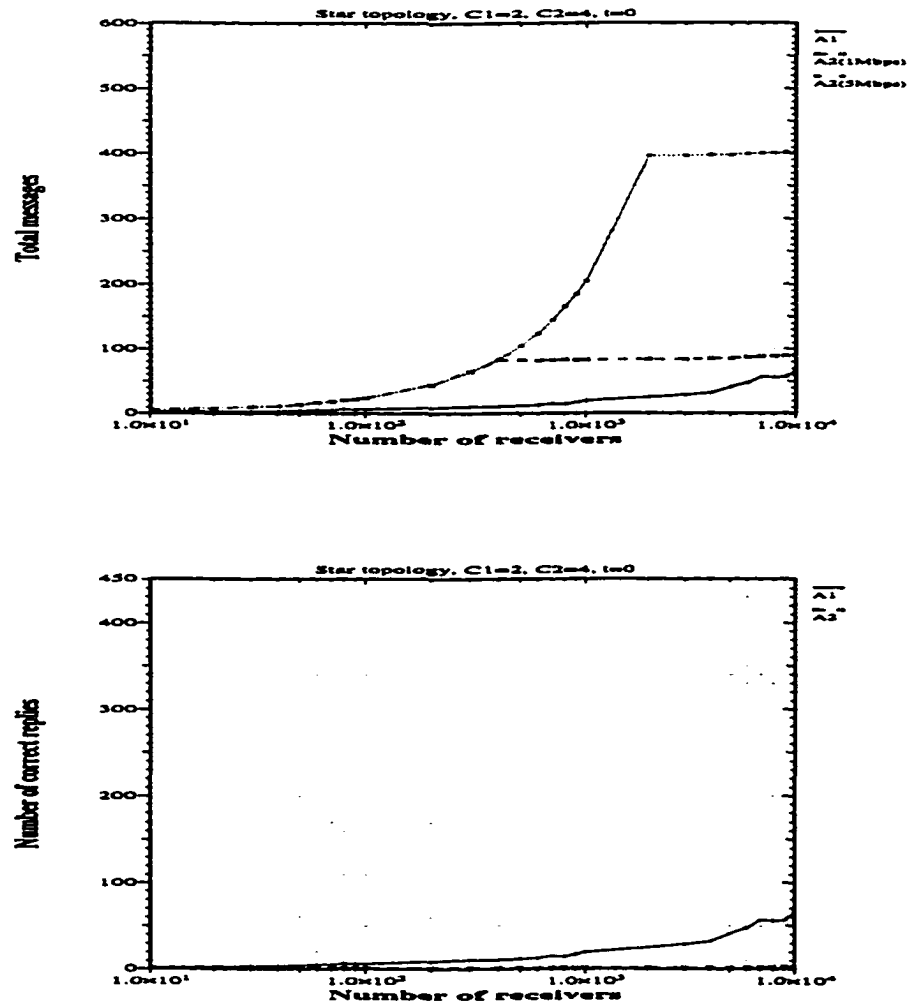


Fig. 4.8. Performance of the star topology for  $t=0$ .

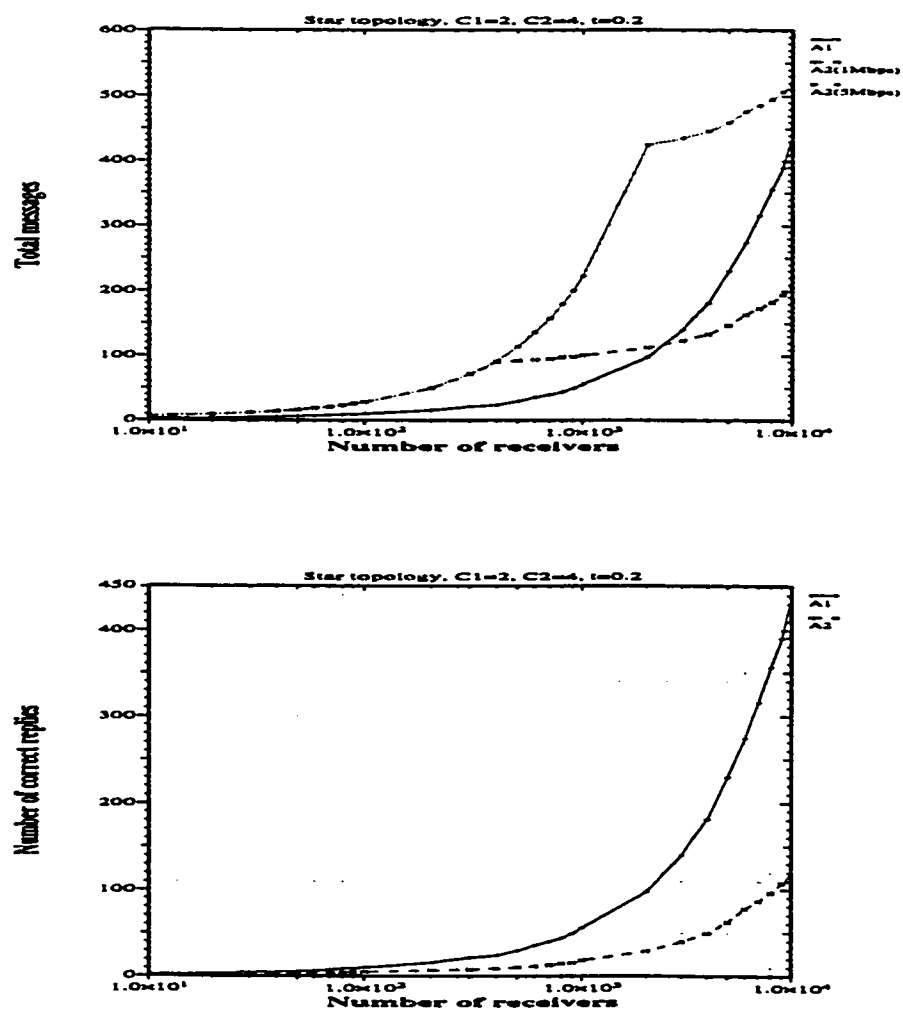


Fig. 4.9. Performance of the star topology for  $t=0.2$ .

depict the two cases of 1Mbps and 5Mbps sessions. For  $A_2$ , the session overhead assumed that an epoch (the time span from sending a probe until receiving the last possible reply) will take at most one second. This should be considered as a best case scenario for  $A_2$ , as round-trip times of over 500 msec are not unlikely over wide area networks.

The figures also show that the number of messages carrying correct worst case state information constitute almost all the total messages sent in the new algorithm  $A_1$ . In  $A_2$ , on the contrary, almost all the messages sent are overhead messages. This demonstrates the robustness of the new feedback mechanism and its tolerance to losses in the network.

However, the figures show that the response time of  $A_2$  is lower on the average. Nevertheless, this is not always the case for  $A_2$ , as a slight shift in the distribution of receiver distances reverses this situation and makes the response time of  $A_1$  lower. This is clear from the graphs in Figure 4.7, which shows the performance of the two algorithms when  $t = 0.2$ . This trend continues as  $t$  increases.

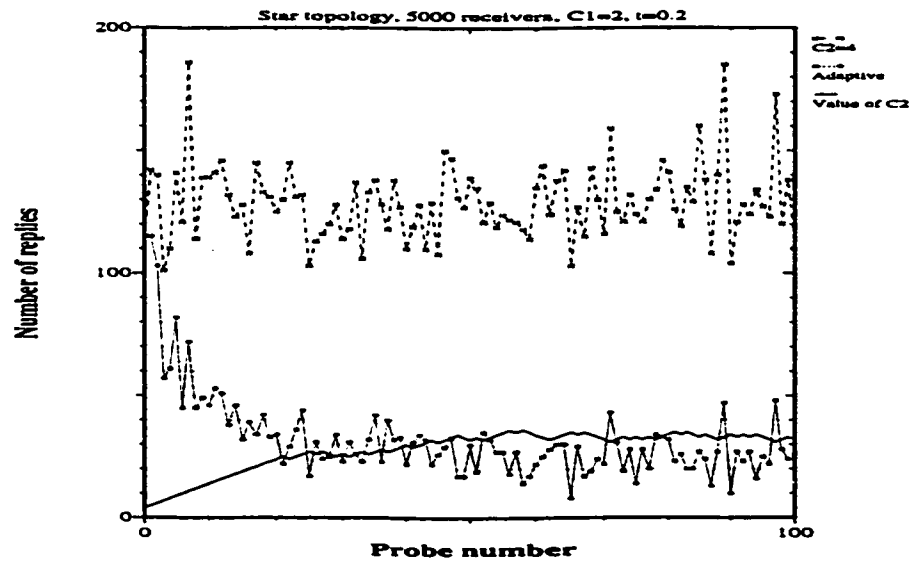
Figure 4.8 and Figure 4.9 depict the behavior of the two algorithms for the star topology. The response time behavior for the star topology is identical to the chain topology, because the distribution of the receiver distances is identical in both cases. The total messages and number of correct replies are different though. From these graphs, we conclude that  $A_1$  is much more robust than  $A_2$ . Also, the total overhead of  $A_1$  is always lower than that of  $A_2$  up to sessions of few thousand participants. However, for very large sessions, approaching 10000 participants, and for certain distributions of distances of receivers, the overhead of  $A_1$  starts to rise significantly. In the next section we address the issue of enhancing the performance of  $A_1$  for very large sessions, and degenerate receiver distributions.

## 4.4 Enhancing the Feedback Mechanism

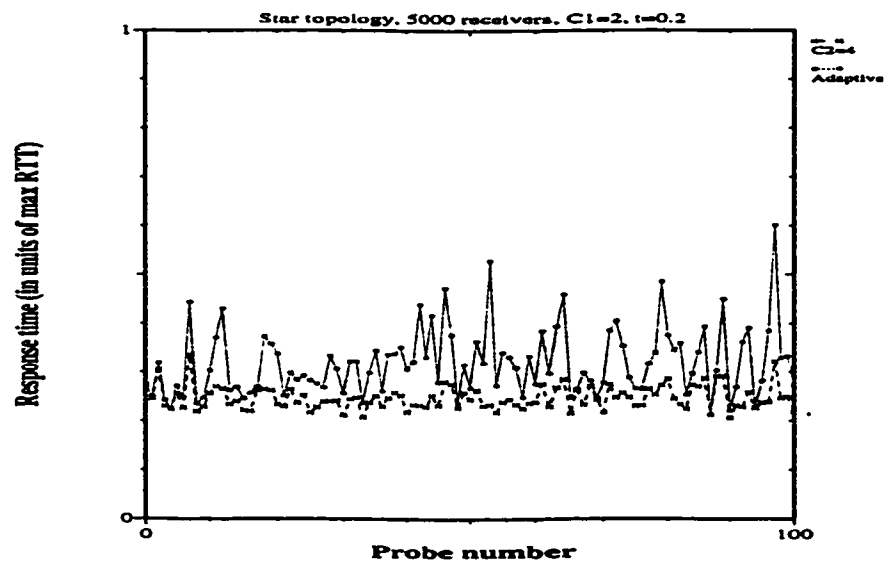
In this section, we present two enhancements for the feedback protocol. These enhancements improve the scalability and reduce the overhead of the protocol.

### 4.4.1 Adaptive feedback

In the previous section, it was shown that the performance of the proposed feedback mechanism needs some tuning to enhance its scalability for very large groups especially in the case when the worst state receivers are far from the sender, and most importantly far from each other. We focus on the worst state receivers because the outcome of the simulation experiments, discussed in the previous section, shows that almost all the excess replies that are generated in these cases are redundant worst case replies. This means that the shift in the start time of the timeout periods is still effective in eliminating replies from lower state receivers. Thus the parameter  $C_1$  does not need tuning. It is the parameter  $C_2$  which needs to be adapted to support very large groups. In other words, as the group size increases too much, the fixed value of  $C_2 = 4$  no longer suffices to effectively suppress enough redundant replies. To this end we developed a simple adaptive algorithm that the sender uses to adapt the value of  $C_2$  dynamically based on the number of received redundant replies. The sender counts the number of redundant worst state replies in response to a probe in the variable *dups*. Note that based on our previous results, the sender can safely count all replies coming in response to a probe assuming they are all worst state replies. Before sending a probe, the sender computes a new value for  $C_2$  and appends it to the probe message. This value is used by the receivers in computing their random timeout periods. The algorithm executed by the sender is given below.



(a)



(b)

Fig. 4.10. Effect of adaptive feedback. (a) Number of replies. (b) Response time.

```

AvgDups =  $\alpha$  AvgDups + (1 -  $\alpha$ ) dups;
if (AvgDups > Threshold)
     $C_2$  = Min( $C_2+1$ , Max_C2);
else
     $C_2$  = Max( $C_2-1$ , Min_C2);

```

Figures 4.10(a) and (b) compare the performance of the static and adaptive feedback. In this simulation experiment, *Min\_C2*, *Max\_C2*, *Threshold*, and  $\alpha$  were set to 4, 50, 25, and 0 respectively. The figures show the ability of the simple adaptive algorithm to reduce the number of redundant replies drastically, without significant delay in response time. The tradeoff, however, is that it takes the sender a longer time before it can declare that the current epoch is over and no further replies will be received. Typically, the sender sends a new probe only at the end of an epoch, to avoid overlapping replies. The sender can always safely terminate an epoch after an amount of time equal to  $(C_1 f(h) + C_2 g(h) + 2) \frac{RTT}{2}$  from sending a probe, where  $h$  is the highest state received in a reply to the current probe. After sending a probe, the sender sets a timer to expire after *RTT* plus the longest possible timeout period in the lowest state, for ending the epoch. As it receives replies, it adjusts this timer according to the above equation which is linearly proportional to  $C_2$ .

A more aggressive approach for ending an epoch without relying on  $C_2$  would be to terminate the epoch after a period of time equal to *RTT* from the time of receiving the first reply. This aggressive approach safely assumes that any reply is coming from the highest state in the group. It attempts to give enough time for this reply to propagate to all other receivers and cause them to suppress their replies, if they haven't already sent it. The approach relies on the heuristic assumption that  $RTT \cong \frac{RTT_{max}}{2}$ :

If it is desired to limit the bandwidth taken by the reply packets to  $R$ , then the *Threshold* value can be set as a function of  $R$ . A simple approach is to set  $Threshold = \frac{R}{Reply\ size} \times Epoch\ duration$ .

#### 4.4.2 Passive feedback

The feedback mechanism, as described, keeps polling the receivers all the time. As soon as the sender determines that an epoch has ended, it immediately sends the next probe. While these probes are important for synchronizing the operation of the mechanism and avoiding potential spontaneous chains of status change notifications from receivers, yet in situations where the states of the receivers are stable for relatively long periods of time, this repeated probing is unnecessary.

One possible solution to optimize the performance of the feedback mechanism in such cases is to make the sender exploit the flexibility in spacing the probes, by increasing the idle time between ending an epoch and sending the following probe. However, this approach negatively affects the responsiveness of the feedback mechanism, especially when a change in state occurs after a relatively long stable state.

Another solution is to switch the feedback mechanism into *passive* mode whenever these relatively long stable states occur. When the sender gets similar state feedback from  $n$  consecutive probes, it sends a probe with a *passive flag* set, and carrying the current highest state  $h$ . Receivers do not respond to this probe, and the sender enters a passive non-probing mode. If a receiver detects that its state has risen above  $h$ , it immediately sets a timer in the usual way to report its state. On receiving a reported new higher state, each receiver updates the value of  $h$ . Similarly, if a highest state receiver detects that its state has fallen below  $h$ , it sets a timer in the usual way. However, when the receivers hear a report below  $h$  they do not update the value of  $h$  (as other receivers may be still in the  $h$  state). On receiving this report, the sender switches back to the active probing mode, and the same cycle repeats.



## 4.5 Conclusions

In this chapter, we presented a scalable and robust feedback mechanism for supporting adaptive multimedia multicast systems. Providing the source of a stream with feedback information about the used layers of the stream is crucial for the efficient utilization of the available resources. The feedback mechanism allows the sender to always send only layers for which interested receivers exist, and to suppress unused layers.

Simulation results showed that the proposed feedback mechanism scales well for groups of up to thousands of participants. For typical sessions with up to 100 participants (e.g., IRI sessions [45]), less than 10% of the receivers reply to a probe, in the worst case, while for larger sessions, of a few thousands of participants, the reply ratio is below 1.5%. The response time was found to be always below the maximum round-trip time from the sender to any of the group members.

The mechanism was shown to be robust in facing network losses, and to be more efficient than mechanisms which rely on session level messages for estimating individual round-trip times from each receiver to the sender. In addition, adaptive enhancements for supporting groups of up to 10,000 participants were proposed and shown to be effective in reducing the number of replies without a significant effect on response time.

In the next chapter, we devise an architecture for realizing the quality of session framework, and which incorporates the presented feedback protocol as one of its main components.

## CHAPTER V

# ARCHITECTURAL DESIGN, PROTOTYPE AND EXPERIMENTAL RESULTS

This chapter discusses the software architecture, and the reference implementation of a *Quality of Session* control layer, that implements the mechanisms presented in the previous two chapters. The *Quality of Session* control layer is designed as a platform for supporting collaborative applications that employ multiple multimedia streams over heterogeneous network and receiver capacities. It manages the bandwidth available to the multimedia session in a scalable and adaptive way. Scalability is achieved by deploying a receiver oriented architecture, in which agents associated with a receiver are responsible for taking decisions on behalf of that receiver only. In this way, heterogeneity of receivers and network connections is dealt with, on behalf of the application, in a distributed and scalable manner, while avoiding any potential conflicting resource allocation decisions.

The *Quality of Session* control layer is composed of two main components: an end-to-end monitoring component, and an inter-stream adaptation component. A monitoring agent is associated with each sender/receiver process. It is responsible for measuring the QoS actually offered to the stream, and for executing a scalable state feedback protocol. An inter-stream adaptation agent runs as a daemon on each receiver machine. It executes a bandwidth allocation mechanism and receiver-based rate control techniques to dynamically control the bandwidth shares of the received streams, in a way that stems from the semantic requirements of the application.

Section 5.1 discusses the principles guiding the design of the QoSess control layer and gives an overview of its software architecture. In Section 5.2, the design details of the monitoring and ISA agents are described. Section 5.3 discusses the used rate control techniques and addresses the stability provisions incorporated into the ISA agents. Our approach to rate control is contrasted to others in Section 5.4. The QoSess layer was prototyped, and Section 5.5 presents results from experiments conducted using the prototype system. Finally, Section 5.6 concludes the chapter.

## 5.1 Design Principles and Architecture Overview

Several design principles guided the process of devising the architecture of the QoSess layer; these are listed below.

**Receiver autonomy.** The QoSess layer is designed based on a receiver oriented approach. An ISA agent is responsible for all the inter-stream adaptation decisions for a single host. As mentioned earlier, the framework advocates multi-grade service streams. This allows for receiver independence and autonomy which eases scaling and accommodation of heterogeneous receivers and network capacities.

**Application level framing (ALF).** The design of the QoSess layer conforms to the concept of ALF [18], which states that the best way to meet diverse application needs is to provide the minimal common functionality, leaving as much flexibility as possible to the application. In the proposed framework, the QoSess layer only dictates the operating points of the streams, while leaving up to the application the structuring of the streams into layers, and the actual adaptation process, which may involve changing some of the encoding parameters.

**Supporting a spectrum of stream types.** Although the proposed framework focuses primarily on hierarchically encoded streams for providing a scalable

multi-grade service, yet the design is general enough to accommodate the co-existence of simulcast and single-rate streams besides hierarchical streams.

**Minimal dependency on traffic characteristics.** The architecture of the QoSess layer is independent of the used traffic characterization model and QoS specification parameters. Also, the parameters monitored by the monitoring agents are general enough to cover a wide spectrum of QoS requirements. However, the algorithms implemented by the decision making unit may vary according to the used traffic specification model.

**Responsiveness and stability.** The QoSess control layer must not react immediately to every detected slight change in the perceived quality of a stream, in order to avoid over-reactions that may lead to instabilities. In the same time, excessive delays in reaction time affect the responsiveness of the system and are not desired. A protocol state machine controls the state transitions of the ISA agent, in a way that ensures stability and responsiveness. This protocol state machine and other stability provisions are detailed in Section 5.3.

**Minimal monitoring overhead.** The monitoring agents are embedded inside the sender and receiver processes. This is primarily intended for eliminating the need for copying the data streams between the agents and their clients. This minimizes the monitoring agent overhead to simply extracting or adding information to the message headers. However, this monitoring efficiency comes at a cost which is an increase in the level of complexity in implementing the monitoring agent, and the need to invoke the agent's API (*Application Programming Interface*) functions from within its client.

The proposed framework relies on the ability of the application to adapt, i.e., the ability to dynamically change the rates of the streams. In addition, in order to support heterogeneity of receivers and network connections, multi-grade streams are centric to the framework. Given the state-of-the-art of the multimedia encoding

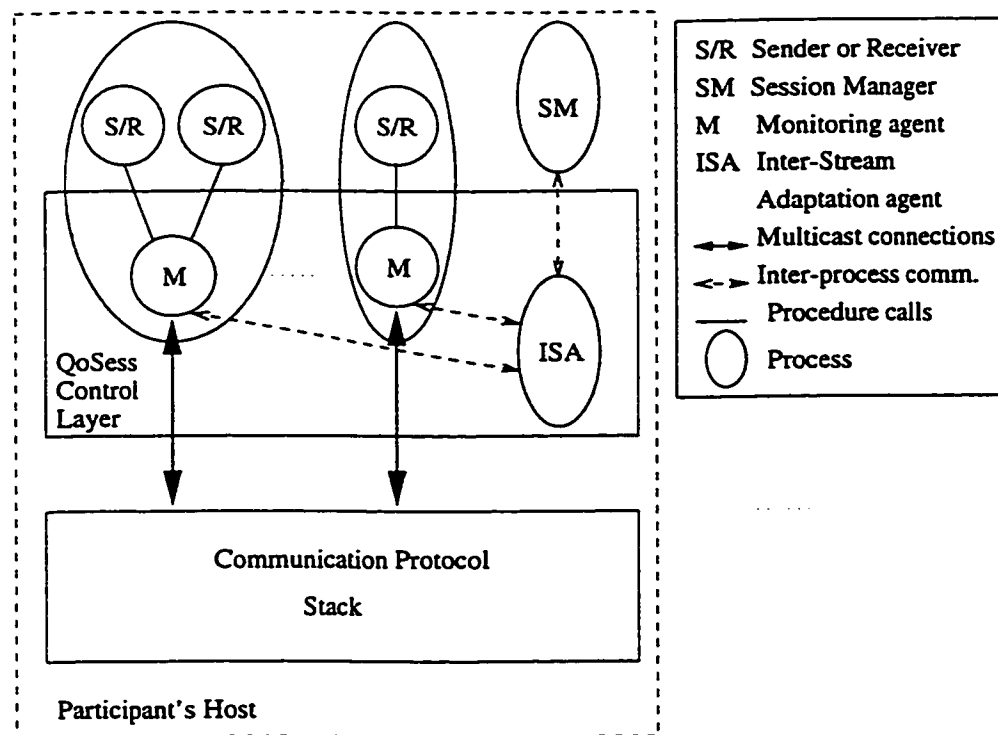


Fig. 5.1. Architecture of the Quality of Session control layer.

techniques, this is not considered, by any means, a stringent constraint [47, 52, 59]. Multi-grade transmission can be achieved either by hierarchical encoding [47, 52], or by simulcast which is the parallel transmission of several streams each carrying the same information encoded at a different grade [42, 59].

Figure 5.1 illustrates the software components of the QoSess control layer. The layer is not a monolithic unit which is embedded in the communication stack. Instead, it is composed of several independent agents that cooperate together to provide the QoSess control framework. Two types of agents constitute the QoSess layer: monitoring agents, and inter-stream adaptation (ISA) agents. A monitoring agent is associated with each sender/receiver process. While several monitoring agents may co-exist on the same host, only one ISA agent runs on each host, as a stand-alone process.

As illustrated in Figure 5.1, the QoSess layer client is modeled in a general abstract form. The client model is composed of sender/receiver units and a session manager (SM) unit. More than a single sender or receiver may be combined together in one process, and supported by a single monitoring agent. No specific requirements are imposed on the architecture of the SM unit. All what is enforced is the type and format of the interface messages. The SM unit itself can be distributed, centralized, or even embedded inside the sender/receiver units. The abstract session manager (SM) unit is responsible for providing the ISA agent with application-specific semantic information, such as the relative priorities of the streams, and the state of each stream, whether it is active or inactive.

## 5.2 Design Details

In this section, we present the design of the different components of the QoSess layer.

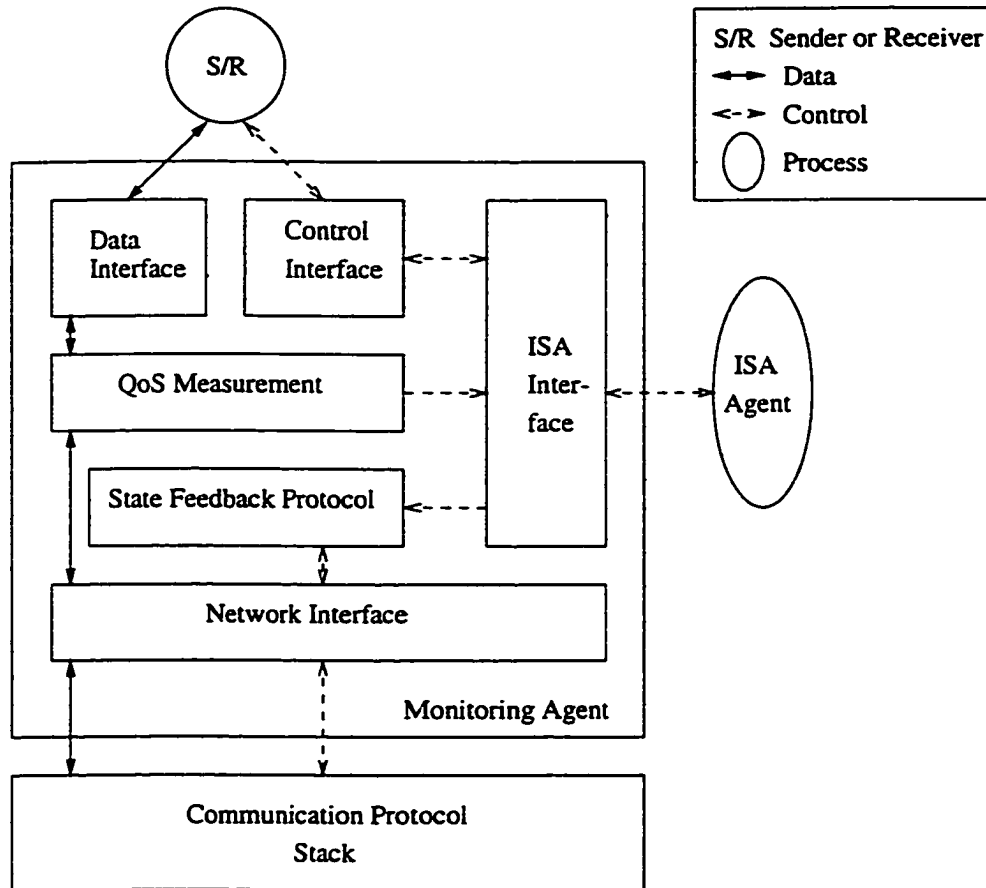


Fig. 5.2. Components of the monitoring agent.

### 5.2.1 Monitoring agents

As illustrated in Figure 5.2, the monitoring agent is composed of several functional units, which are described here focusing on the interfaces, especially the API used by the clients. As previously mentioned, the monitoring agent is provided as a library which is linked with the client code at compilation time.

#### Client interface

The API available to the QoSEss layer clients is divided into two main interfaces: a data interface, and a control interface. The data interface is used for sending (receiving) the data packets. This enables the agent to add (extract) QoSEss headers.

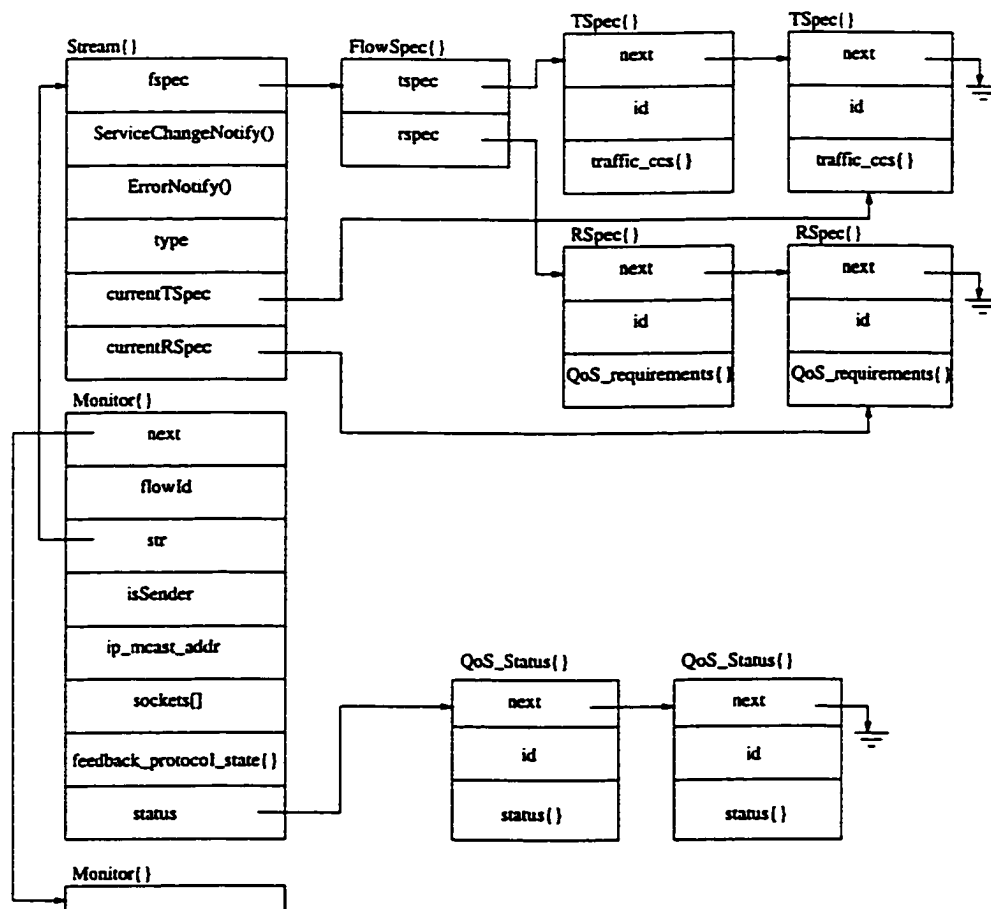


Fig. 5.3. The Stream and Monitor data structures.



These headers contain the information necessary to allow the agent for computing the actual QoS offered to the stream.

A typical concern whenever a layer is added to the communications protocol stack, is the overhead incurred with the addition of this layer. In order to minimize the monitoring overhead, no data copying is done at the client data interface. As well known, data copying is the major source of overhead at adjacent layers in the communication stack. This reduces the introduced overhead, for the data path, to simple header information addition or extraction. On the sender's side, the sending client requests QoSess buffers through the API. The agent allocates the requested memory in addition to the header size and returns a pointer to the data portion of the buffer to the sender. When the sender invokes the agent's send routines, the header fields in the buffer are filled in before sending it out. Again no data copying is made at this stage. On the receiver's side, the header information is extracted from the received packet and a pointer to the data portion is returned to the receiving client.

Through the control interface, the client provides the agent with information regarding the stream's characteristics, at initialization time. The agent communicates this information to the local ISA agent, which uses it in making rate adaptation decisions. On receiving an ISA decision regarding the operating point of the stream, the agent joins (leaves) the appropriate multicast groups on behalf of the client, then triggers a callback function to inform its client about the new operating point. In addition, the control API provides the client with wrappers for the *alarm()* and *select()* system calls. Each of these wrappers either export to the client the same interface of the original call or enhance it, while allowing the agent to setup timers and multiplex its own input sockets appropriately. The API calls of the data and control interfaces are detailed in Appendix B.

Two main data structures are of particular importance to the client. Figure 5.3 illustrates the *Stream* and the *Monitor* structures. The client must be aware of the details of the Stream structure, as it is the client's responsibility to provide

the monitoring agent with an initialized Stream structure (except for the currently selected operating point which is updated by the agent). It should be noted however that the use of profiles, as described in Appendix B, can extremely simplify this task. The components of the Stream structure are listed below.

**fspec** holds the flow specification which points to a list of traffic characteristics and another list of QoS requirements. Each node in the traffic characterization list contains the characteristics of one layer, with the first layer (with id=0) being the base layer for the stream. The actual parameters used to characterize the traffic depend on the model used. For example, for constant bit rate (CBR) or smoothed sources, the bit rate of the stream is the only parameter. Another example is the four parameters of the M-LBAP traffic characterization model described in Section 3.4. The QoS requirements represent the desired behavior from the network, in terms of maximum tolerated delay, jitter, and losses. The rate requirements are provided implicitly in the traffic characteristics. The QoS requirements list can be either composed of one node only or a node corresponding to each traffic characteristics node.

**ServiceChangeNotify()** is a handler provided to the monitoring agent. When a decision is made by the QoSess layer to change the operating point of the stream, this handler is invoked to provide the client with the new operating point. The operating point is defined by two pointers; one pointing at the new traffic characteristics node, and another pointing at the new QoS requirements node.

**ErrorNotify()** is a handler for notifying the clients about severe errors detected by the QoSess layer.

**type** identifies the nature of the stream, whether it is a layered, simulcast or single-rate stream.

**currentTSpec** and **currentRSpec** are two pointers to indicate the operating point currently selected by the ISA agent. Whenever any of these pointers is modified by the monitoring agent, *ServiceChangeNotify* is invoked.

The *Monitor* structure, on the other hand, is not modified by the client. During initialization, the client obtains a reference to its Monitor which it uses in accessing the API functions. The components of the structure are described below.

**next** is a pointer to the next Monitor supported by this agent, if exists.

**flowId** is the unique identifier of the stream. This is typically the UDP port used for the base layer of the stream.

**str** is a pointer to the associated Stream structure.

**isSender** a flag to differentiate whether the client is a sender or a receiver. This mainly affects the feedback protocol operation. Note that multiple sender and receiver clients may be supported by a single agent.

**ip\_mcast\_addr** is the base layer multicast group address. Addresses of other layers are assumed to be derivable from this base address by a simple formula.

**sockets[]** is an array of sockets; a control socket for feedback and exchange of control information, and a data socket for each layer.

**feedback\_protocol\_state{}** maintains the state information related to the feedback protocol described in Chapter 4.

**status** is the head of a list of nodes; one node corresponds to each layer of the stream and maintains the current and cumulative measured QoS parameters for that layer.

TABLE 5.1  
MONITOR/ISA INTERFACE MESSAGES

Name	Fields	Description
MtoISA_FlowSpec	flowId, type, isSender, FlowSpec{}	Inform ISA agent about the specifications of a new stream.
MtoISA_StreamQuality	flowId, quality	Report current stream quality to ISA agent.  quality $\in$ { <i>Underqualified</i> , <i>Acceptable</i> , <i>Overqualified</i> }.
ISAtom_OpPoint	flowId, TSpecId, RSpecId	ISA agent instructs monitor about selected operating point.

### ISA interface

Table 5.1 describes the messages exchanged between the monitoring agent and the ISA agent running on the same machine. During initialization, the monitoring agent communicates the flow specification of the stream to the ISA agent, using the *MtoISA\_FlowSpec* message. Periodically, the monitoring agent sends *MtoISA\_StreamQuality* reports, which the ISA agent uses in assessing the overall quality of the session from the perspective of this host. This in turn may trigger *ISAtom\_OpPoint* notifications of the newly selected operating point for the stream.

### Network interface

The monitoring agent sends/receives data on behalf of the client. Also, it exchanges state feedback protocol messages with other monitoring agents supporting clients for the same stream. The agent uses system calls of the socket layer API [60] to interface to the network. In UNIX-based implementations, the same system calls are used by the agent to communicate with the ISA agent running on the same machine, by means of UNIX domain sockets [54].

### State feedback protocol

In Chapter 4, we presented a scalable and robust feedback protocol which provides the sender of a multimedia stream with deterministic information regarding the state of the receivers. Given this knowledge, the sender can take appropriate reactions, based on the nature of the stream. If it is a hierarchically encoded stream, the sender can suppress or start encoding and sending the correct layers, while if it is a single-rate stream, the sender can adjust the transmission rate accordingly. The monitoring agent implements this state feedback protocol, and executes a protocol machine for each stream supported by the agent. The state input to the protocol machine is the current operating point selected by the ISA agent.

### QoS measurement

QoS measurement is a fundamental purpose of the monitoring agent. The measured stream quality is fed to the ISA agent in order to be accounted for in the adaptation process. Let  $Q$  be the set of all possible quality grades that can characterize a stream. The monitoring agent determines the perceived quality of a stream and maps it to one of the grades in  $Q$ . We define the set  $Q$  as:

$$Q = \{Underqualified, Acceptable, Overqualified\}.$$

Typically one or more of the following factors are monitored to indicate the effective QoS offered: loss ratio, inter-arrival jitter, effective throughput, round trip delay (RTT), and delay since last packet received. The delay since receiving the last packet from the source is important in cases when all packets sent by the source, after the last received packet, are lost. Hence this factor helps in the detection of losses in this special situation. The inter-arrival jitter,  $J$ , is defined to be the mean deviation (smoothed absolute value) of the difference,  $D$ , in packet spacing at the receiver compared to the sender for a pair of packets [51]. As shown in the equation below, this is equivalent to the difference in the relative transit time for the two packets. If  $S_i$  is the time-stamp, given at the sender, for packet  $i$ , and  $R_i$  is the time

of arrival of packet  $i$ , measured at the receiver, then for two packets  $i$  and  $j$ ,  $D$  may be expressed as:

$$D(i, j) = (R_j - R_i) - (S_j - S_i) = (R_j - S_j) - (R_i - S_i).$$

The inter-arrival jitter is calculated continuously as each data packet  $i$  is received from the source using the difference  $D$  for that packet and the previous packet  $i - 1$ , according to the formula:

$$J = J + (|D(i - 1, i)| - J)/16.$$

This algorithm is the optimal first-order estimator and the gain parameter  $1/16$  is the optimal noise power reduction ratio for situations where there is no model of the system [51]. The jitter measure is expected to indicate congestion before it leads to packet losses.

In QoS measurement in a multicast setup, it is not desired to rely on factors that require for their correct computation continuous feedback to the sender. RTT is one such factor. Therefore, it is not considered an option for us. Fortunately, our interest lies in RTT variations, as opposed to the actual RTT values. These variations indicate a change in the offered QoS, while long but constant RTT values simply imply the presence of some slow links in the communication path. The inter-arrival jitter captures these variations precisely. In fact it is more accurate than RTT variations since it accounts for variations in the one way delay only from the source to the receiver.

Given knowledge about the inter-arrival jitter at the receiver relative to the sender, together with knowledge about losses, renders throughput measurement at the receiver redundant. Thus, the two main factors on which the receiver's monitoring agent relies in detecting congestion are the packet losses and inter-arrival jitter (hereafter referred to as jitter). Correct measurement of these two factors requires the inclusion of a sender's timestamp and sequence number in the header of each packet. These two fields are part of the standard header of the real time transport

protocol (RTP) [51]. Although the QoSS layer implementation is independent of RTP, and may operate on top of UDP directly, yet in cases where RTP is deployed and implemented at the application level, it is recommended to combine the QoSS layer and RTP implementations into one layer which uses the standard RTP header.

We demonstrate an example for determining the value of  $q$ , the measured quality of service, based on the loss ratio. Let  $L$  be the loss fraction as measured at the end of the current monitoring interval,  $T_r$ . The two threshold values  $L_{min}$  and  $L_{max}$  are defined such that:

$$q = \left\{ \begin{array}{ll} \textit{Overqualified} & \textit{if } L \leq L_{min} \\ \textit{Acceptable} & \textit{if } L_{min} < L \leq L_{max} \\ \textit{Underqualified} & \textit{if } L > L_{max} \end{array} \right\}.$$

$L_{max}$  is set to the maximum loss fraction tolerated as specified in the flow specification.  $L_{min}$  is set to a fraction,  $f$ , of  $L_{max}$ , where  $f \leq 0.5$ , typically.

Each layer of a stream is monitored separately. At the end of a monitoring interval, the quality of the layer is considered *Underqualified* if any of the QoS requirements specified in the flow specification is violated. It is considered *Overqualified* if all the monitored parameters are in the *Overqualified* range. Otherwise it is considered *Acceptable*. In other words, the conjunction of all monitored factors must yield *Overqualified* for the quality of the layer to be considered *Overqualified*. However the disjunction of the monitored factors yielding *Underqualified* is enough to consider the layer quality *Underqualified*. Similarly, the overall quality of the stream is considered *Overqualified* if the quality of each layers is *Overqualified*. It is considered *Underqualified* if the quality of any layers is *Underqualified*. Otherwise, it is considered *Acceptable*.

## 5.2.2 Inter-stream adaptation agents

The structure of the ISA agent is depicted in Figure 5.4. In this subsection, we describe each of the functional units comprising the ISA agent.

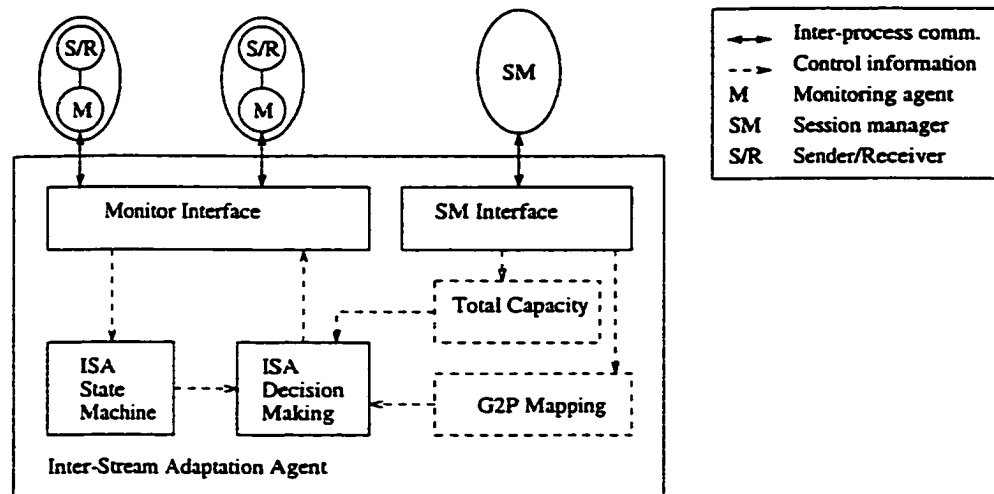


Fig. 5.4. Components of the inter-stream adaptation agent.

### Session manager interface

The ISA agent interfaces to two external modules: the monitoring agent, and the session manager (SM). The ISA/Monitor interface was described in the previous section, while the ISA/SM interface is detailed here. The QoSS layer does not impose any specific requirements on the architecture of the SM. All what is enforced is the type and format of the interface messages. The SM itself can be distributed, centralized, or even embedded inside the sender/receiver module. In order to accommodate the latter case, the monitoring agent client interface must be extended to provide an additional interface function corresponding to each of the SM messages listed in Table 5.2 below. The client invokes the function which in turn causes the monitoring agent to send the corresponding SM message to the ISA agent.

The abstract SM is responsible for providing the ISA agent with information regarding the relative priorities of the streams, and the state of each stream, whether it is active or stopped. Table 5.2 describes the messages which the ISA agent can receive from the SM.

*SMtoISA\_BorrowRelation* is used to construct a QoSS graph, as explained in Section 3.2. The QoSS graph represents the relative priorities of the streams,



TABLE 5.2  
ISA/SM INTERFACE MESSAGES

Name	Fields	Description
SMtoISA_Activate	flowId	Request to activate a stream.
SMtoISA_Deactivate	flowId	Request to stop a stream.
SMtoISA_BorrowRelation	flowId1, flowId2	Set a borrow relationship from flowId1 to flowId2 (ID of slack node is -1).
SMtoISA_Priority	flowId, priority	Set a stream's priority (used only if QoSess graph is disabled).
SMtoISA_Capacity	capacity	Inform ISA agent about session reserved bandwidth (if exists).

as dictated by the application semantics. This representation defines for each node (stream) the set of nodes from which it can borrow resources, and is used by the ISA decision making unit.

### G2P mapping

In Section 3.2, we devised an algorithm, G2P, for mapping a given QoSess graph into a corresponding set of priority classes. This mapping algorithm decouples the inter-stream adaptation policy from the mechanism used to implement it. The application specifies its needs in a flexible way that reflects its semantic requirements without interfering with the parameters that control the adaptation process. It should be noted that the G2P mapping is optional, and can be bypassed if the implemented inter-stream adaptation algorithm uses the QoSess graph directly as input instead of priority classes.

### Capacity information

In situations where a group reservation exists, e.g., when using RSVP [67], or if a special network connection is dedicated for the session, the SM needs to inform the ISA agent about the total reserved bandwidth. This information is useful for the ISA decision making unit as it aids in preventing potential overload situations, and in eliminating unnecessary probing to determine the availability of bandwidth beyond the reserved level. The SMtoISA\_Capacity message is used for this purpose.

### Monitor interface

This module is responsible for processing messages received from the monitoring agents running on the same host, as well as sending messages to those agents. The messages exchanged between ISA and monitoring agents were detailed in Section 5.2.1.

### ISA decision making

This module represents the brains of the ISA agent. It implements the mechanisms necessary to select dynamically the operating point for each stream within the stream's operating range which is given in its flow specification. The ISA decision making unit is triggered to recompute the operating points of the active streams by the ISA protocol state machine, when the latter detects either an overload or an underload overall receiver state. Also, the decision making unit is triggered by external events such as the activation/deactivation of a stream or the change of the relative priorities of some of the streams.

We proposed more than one algorithm for inter-stream adaptation in Chapter 3. The A-IWFS algorithm, presented in Section 3.6, is the one implemented in the reference implementation of the QoSess layer, due to its relative merits.

## ISA state machine

The ISA agent is responsible for dynamically allocating the resource shares for each of the streams belonging to a session. Consequently, the operating points of the streams change. These changes are mostly in response to notifications received from the monitoring agents regarding the perceived quality of the streams. These dynamic changes should be handled carefully in order to avoid instabilities and oscillations in operating points. The ISA state machine controls the state transitions of the ISA agent, in a way that ensures stability and responsiveness. It is detailed, together with other stability provisions, in the following section.

## 5.3 Stability Provisions

In this section, we first describe the operation of the ISA agent, using a state transition diagram. Then, we proceed to elaborate on the control of the parameters that trigger state transitions. Finally, we present a domain rate control protocol [64], which establishes a framework for cooperation and sharing of knowledge about the network state, among ISA agents residing in the same local domain.

### 5.3.1 Inter-stream adaptation state machine

The state transition diagram depicted in Figure 5.5 provides a parameterized mechanism for controlling the performance of the ISA agent. Transition from one state to another is guided by one of two conditions, or by a combination of both. The first condition is the expiration of a timer that is set at the time of entering the state. The second condition, denoted by  $q$ , represents the aggregate state of all streams as described below. At every instant in time, the ISA agent is in one of the following four states.

1. **Overqualified (O) state.** The agent enters this state if the state of all streams, as reported by monitoring agents, is Overqualified ( $q = O$ ). The

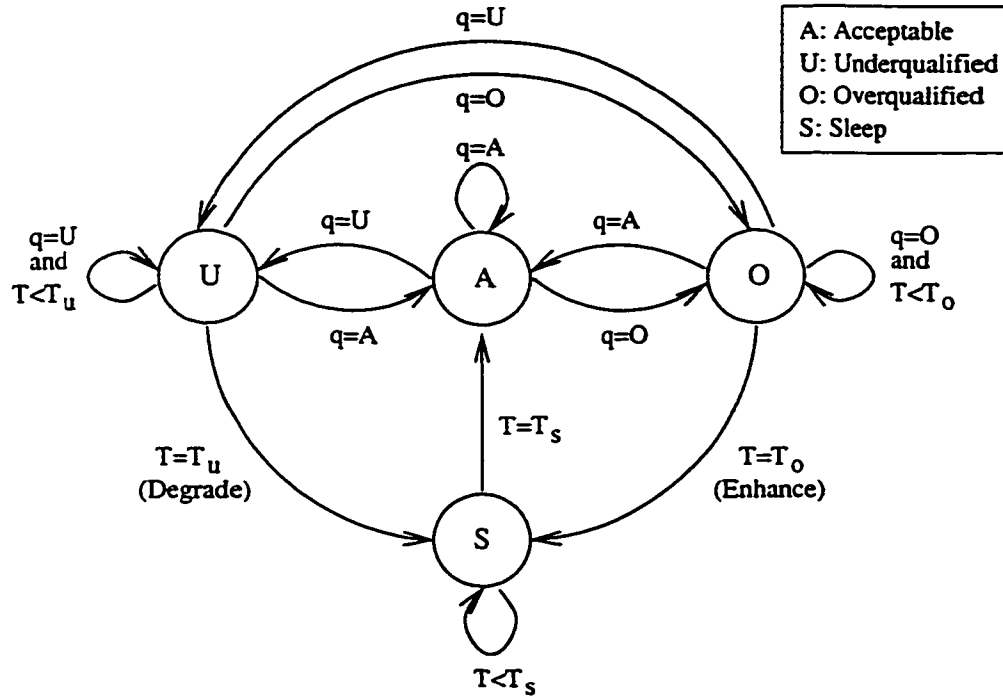


Fig. 5.5. Inter-stream adaptation protocol state machine.

timeout period  $T_o$ , from the time of entering the state till the time of making an enhancement decision, ensures non-reacting to false notifications.

2. **Underqualified (U) state.** The agent enters this state if the state of at least one stream, as reported by monitoring agents, is Underqualified ( $q = U$ ). The timeout period  $T_u$ , from the time of entering the state till the time of making a degradation decision, ensures non-reacting to false congestion signals.
3. **Acceptable (A) state.** The agent is in this state if none of the streams is Underqualified, and not all of them are Overqualified ( $q = A$ ). This state represents the system's stable state.
4. **Sleep (S) state.** The agent enters this state immediately after making an adaptation decision. In this state, no further reactions are taken by the agent for  $T_s$  units of time. This ensures that the previous action is already in effect, thus avoiding multiple unnecessary reactions (over-reacting) to multiple

monitoring reports reflecting the same condition.

The parameters  $T_u$ ,  $T_o$ , and  $T_s$  control the reaction speed of the agent. Tuning these parameters for best performance, while maintaining system stability, is crucial. The following subsections discuss this issue in detail.

### 5.3.2 Multi-modal timers

In this section, we closely examine the role of the two timer parameters  $T_u$  and  $T_o$  in controlling the system reaction time. When congestion is sensed by an ISA agent, it moves into the U state. In order to ensure not reacting to false or transient indications, the system must stay in state U for a period  $T_u$  before any layers are dropped. Similarly, before adding a layer, the system must stay for a period of  $T_o$  in state O. Careful setup of these two parameters is not only important for the stability of the system, but for its responsiveness as well. Setting those parameters to constant values may serve one but not both of the objectives. This is because stability demands long timeouts while responsiveness requires shorter timeouts.

The key to satisfying these two objectives is to be able to accurately capture, at all times, the tendency of the system whether it is towards enhancement if the conditions are favorable, or towards degradation if the network or host are congested, or towards stabilizing at a certain subscription level that reflects the available resources in the network and host. We refer to this tendency in the ISA agent as *mode*. This mode must be reflected on the timers behavior dynamically. The ISA agent can be in one of three modes at any instant in time; these are enumerated below.

1. **Enhance mode.** Being in this mode means that the overall conditions have been favorable for some time and the tendency of the agent is towards adding more layers, thus  $T_o$  is relaxed. Also, any intermittent variations of the monitored QoS parameters are more likely to be transient thus  $T_u$  is backed off to ensure reactions only in case of persistent QoS degradations. In this mode, the

timer parameters are updated after each add/drop action taken by the agent as follows.

$$T_o = \text{Max} \left( \frac{T_o}{\alpha_o}, T_o^{\min} \right)$$

$$T_u = \text{Min} (\alpha_u T_u, T_u^{\max})$$

where,  $\alpha_o$  and  $\alpha_u$  are the back off/relaxation factors of  $T_o$  and  $T_u$ , respectively.  $T_o^{\min}$  is the minimum allowed value for  $T_o$ , and  $T_u^{\max}$  is the maximum possible value for  $T_u$ .

2. **Degrade mode.** Being in this mode means that the receiver has been congested for some time and the tendency of the agent is towards dropping layers to relieve the congestion. In this mode,  $T_u$  is relaxed to increase the reaction speed of the system, while  $T_o$  is backed off to ensure that detection of any transient favorable conditions does not make the agent oscillate between adding and dropping layers. In this mode, the timer parameters are updated after each add/drop action taken by the agent as follows.

$$T_u = \text{Max} \left( \frac{T_u}{\alpha_u}, T_u^{\min} \right)$$

$$T_o = \text{Min} (\alpha_o T_o, T_o^{\max})$$

where,  $\alpha_o$  and  $\alpha_u$  are the back off/relaxation factors of  $T_o$  and  $T_u$ , respectively.  $T_u^{\min}$  is the minimum allowed value for  $T_u$ , and  $T_o^{\max}$  is the maximum possible value for  $T_o$  in this mode.

3. **Probe mode.** In this mode, the agent is stabilizing around an operating point that reflects the current available resources to the session streams. The agent has to keep probing periodically to check for the availability of more resources, as long as it did not hit the capacity limit specified to the agent or if it does not know about that limit. This probing is done by adding a layer, and examining the effect of joining this layer on the system performance. If any deterioration in the measured QoS parameters is noticed, the layer is immediately dropped.

This is called a *join experiment* or simply a *probe*. Here, it is important to back off the  $T_o$  parameter over time to relatively larger values in order to minimize the number of transient disturbances introduced by the probes. In this mode, the  $T_o$  timer parameter is backed off after each drop action taken by the agent as follows.

$$T_o = \begin{cases} \text{Min}(\alpha_p T_o, T_p^{max}) & \text{if in Probe mode already} \\ T_p^{min} & \text{if entering Probe mode} \end{cases}$$

where,  $T_p^{min} > T_o^{max}$  typically, and  $T_p^{max} \gg T_p^{min}$ .

Also in this mode, when a layer is dropped,  $T_u$  is set to its maximum value  $T_u^{max}$ , to prevent any over reaction to transient degradations in QoS due to probing, which helps in maintaining the stability of the system.

It should be noted that  $T_o$  is not backed off, in this mode, after an add action to prevent two back offs happening at almost the same time (one at the add and one at the following drop action). However, if adding the layer does not cause trouble, the join experiment succeeds, and the agent moves to the Enhance mode, where the timers are immediately updated according to the enhance mode rules stated above.

Whether in Enhance or in Probe mode, at the time of joining a layer,  $T_u$  is set temporarily to  $T_u^{join}$  such that  $T_u^{join} < T_u^{min}$ . This ensures that the agent performing the join experiment will be the first to detect its negative effects and react to them (by dropping the layer) quickly enough before any other agents in the domain which might be affected by congestion introduced by joining this layer. The value of  $T_u$  is restored to its original value when the join experiment is over. The experiment is considered to be over either after a period of  $T_s + T_r + T_u^{min}$  from its start, where  $T_r$  is the reporting interval from the monitoring agents, or at an earlier time when a decision to drop the newly added layer is made. In the steady state, the agent needs at most a period of  $T_s + T_r + T_u^{join}$  from joining a layer to detect the failure of the join experiment, as will be explained in the next section. Hence,

$T_s + T_r + T_u^{min}$  is enough time to determine that the experiment is successful because  $T_u^{join} < T_u^{min}$ .

In order to determine the mode of the ISA agent, we follow a heuristic approach based on the most recent two actions taken by the agent, as shown in Table 5.3.

TABLE 5.3  
HEURISTICS FOR DETERMINING THE MODE OF AN ISA AGENT

Previous Action	Current Action	Mode
Drop	Drop	Degrade
Add	Add	Enhance
Drop	Add	if succeed then Enhance else Probe
Add	Drop	Probe

### 5.3.3 Learning network delay

In this section, we closely examine the role of the timer parameter  $T_s$  in ensuring the stability of the system. After adding or dropping a layer in reaction to detected network conditions, the ISA agent must not take any further actions until it is sure that the impact of its previous action on the network is fully established and can be detected by itself, and hence the currently seen conditions are correct so it can make correct decisions. According to the state transition diagram in Figure 5.5, after taking an action, the agent is guaranteed not to take any further actions for a period  $T_s$ . Therefore,  $T_s$  must be a good indicator of the network reaction time. This is done by measuring the time that elapses from adding a layer until congestion is first detected by the agent, in a failed join experiment. The value of  $T_s$  is smoothly updated over time by these measured periods, using the commonly used technique of *exponential weighted moving average* [40].



A more conservative approach would be to set  $T_s$  to the sum of two components; one of which is a factor of the smoothed mean sample deviation, and the other is a factor of the smoothed average. This is the mechanism used by TCP in estimating the round trip delay [40]. However, as we have shown before, this amount is too conservative and over estimates the network delay in a way that may affect the responsiveness of the agent.

It should be noted that in the S state, the agent is sleeping with respect to add/drop actions and making transitions to other states only, yet it still receives monitor reports and is able to know when congestion is first sensed, i.e., the value of  $T_s$  may increase or decrease over time according to the network delay.

#### 5.3.4 Domain rate control protocol

The objective of the domain rate control protocol is to help in maintaining the stability of the system while scaling to large groups of participants in a session. There is no doubt that the receiver oriented approach taken, where each participant decides for himself which layers of which streams to receive, is the key for scalability. Moreover, this receiver oriented approach is what allows each ISA agent to employ techniques such as multi-modal timers and learning network reaction time from its own perspective to achieve stability at the controlled host. However, the co-existence of several ISA agents in the same session opens further avenues for cooperation among those agents to enhance the stability of the system.

Our approach is to group the ISA agents of a session into domains. A domain is defined by the scope of the exchanged protocol control messages as determined by the time-to-live (TTL) field specified in those messages. Typically a TTL of one, or a subnet, will be considered as a domain. However, other values are possible and the subsequent discussion is independent of a specific TTL choice, although TTL values above eight are not considered as an option, practically.

One avenue for enhancing the stability of the system is to minimize the

number of probes to higher layers above the current stable level, by letting other ISA agents in the same domain to learn about *join experiments* performed and their results. Learning about failed join experiments allows the other agents to back off their timers and update their estimators for network reaction time without actually probing, thus minimizing instabilities caused by overload from such probes. The fact that the scope of the domain is very limited is what allows for safely assuming that all agents in the domain are likely to face similar conditions when they probe.

Also, cooperation among ISA agents in the same local domain is useful in preventing unnecessary oscillations in the subscription levels of low rate receivers in the domain. In the case of network overload conditions, letting higher level subscribers drop their upper layers first may be sufficient to reduce congestion in the domain. Thus, coordinating the reactions of the ISA agents in a domain will yield better stability for low rate receivers which would otherwise react to the load unnecessarily by dropping layers which they find themselves immediately capable of re-joining again. In what follows, we describe our proposed domain rate control protocol.

### Protocol state variables

Each ISA agent maintains the following protocol state variables:

1.  $R_c$ . This variable maintains the current total rate of all layers from all streams that are currently activated by the ISA agent. This rate is computed based on the average rate in the streams' specifications.
2.  $R_h$ . This variable maintains the maximum current  $R_c$  in the domain.

### Protocol messages

The following types of messages are exchanged between the ISA agents that reside in one domain:

1. **ADD( $R_h$ )**. When an ISA agent decides to probe for the next layer up, and finds out that its  $R_c$  will be greater than the current  $R_h$ , it multicasts an ADD message to all other ISA agents in the domain, containing the new highest rate in the domain after the subscription is completed. The following pseudo code explains the actions taken by an ISA agent at the time of adding a layer.

$R_l$  = rate of layer to add ;

$R_c = R_c + R_l$  ;

if (  $R_c > R_h$  ) {

$R_h = R_c$  ;

send ADD( $R_c$ ) ;

}

2. **DROP\_REQ( $R_c$ )**. When an ISA agent who currently has the highest level of subscription ( $R_h$ ) decides to drop one layer, it multicasts a DROP\_REQ message to all other ISA agents in the domain. The objective of this message is to solicit confirmation/denial of network congestion from other  $R_h$  agents if any exists. If only one other  $R_h$  agent detects a similar condition, that agent will acknowledge the congestion implying a mandatory degradation for all other  $R_h$  agents. However, if all other  $R_h$  agents negatively acknowledge the congestion, this means that the condition is local to the host who detected it and  $R_h$  is not changed. If no replies are received, the agent concludes that it is the only  $R_h$  agent and it sends a DROP\_ACK message itself to announce the new  $R_h$  in the domain. The following pseudo code explains the actions taken by an ISA agent at the time of dropping a layer.

$R_l$  = rate of layer to drop;

$R_c = R_c - R_l$  ;

if (  $R_c + R_l == R_h$  ) {

send DROP\_REQ( $R_c$ ) ;

schedule a DROP\_ACK message after  $\delta$  seconds ;

```

    NAK_Received = False ;
}

```

3. **DROP\_ACK( $R_h$ ,  $T_s$ )**. On receiving a DROP\_REQ message, an ISA agent which is subscribing to the highest rate in the domain, and which is currently in the U state, sends a DROP\_ACK message. This message announces the degradation of the highest rate in the domain, and lets the lower rate agents realize that some layers were dropped so they delay further reactions until after the effect of reducing the upper layers on the domain are in place.  $R_h$  is the new highest rate in the domain, and  $T_s$  is the updated sleep timeout parameter described in the previous section.
4. **DROP\_NAK( $R_h$ )**. On receiving a DROP\_REQ message, an ISA agent which is subscribing to the highest rate in the domain, and which does not detect any congestion developing in the domain, sends this message containing a re-confirmation of its highest rate.

### Handling ADD messages

On receiving a message ADD( $R$ ), the ISA agent updates  $R_h$  and goes to state S (Sleep). This ensures that no join experiments will be performed in parallel since the outcome of such experiments, if performed, will be ambiguous because any congestion happening in this time will most likely be due to the new traffic entering the domain due to the new highest layer subscription. On the other hand, if this agent is anticipated to perform a similar experiment in the near future, it starts the experiment instantaneously. This minimizes the number of independent probes in the domain. The following pseudo code describes the ISA agent's behavior in response to receiving an ADD( $R$ ) message.

```

if ( $R > R_h$ ) {
     $R_h = R$  ;
     $R_l$  = rate of next layer to add ;
}

```

```

if (  $R_c + R_l == R$  ) AND ( state == O ){
     $R_c = R_c + R_l$  ;
    Add layer ;
}
goto state S ;
}

```

### Handling DROP\_REQ messages

Only the highest rate subscriber ( $R_h$ ) ISA agents react to DROP\_REQ( $R$ ) messages. If the agent is in the U (Underqualified) state, it immediately drops the highest layer and sends a DROP\_ACK message to the agents in the domain. This means that a consensus of two agents on congestion in the domain is assumed sufficient to force drop the highest layer at all its subscribers in the domain. On the other hand, if the agent is not in the U state, it sends a DROP\_NAK message. The following pseudo code describes the ISA agent's behavior in response to receiving a DROP\_REQ( $R$ ) message.

```

if (  $R_c \leq R$  )
    return ;
 $R_l$  = rate of next layer to drop ;
if (  $R < R_h - R_l$  )
    return ;
if ( state == U ) {
     $R_c = R$  ;
     $R_h = R$  ;
    drop layer and make transition to state S ;
    send DROP_ACK( $R_h, T_s$ ) ;
} else
    send DROP_NAK( $R_h$ ) ;

```

### Handling DROP\_ACK messages

On receiving a message  $\text{DROP\_ACK}(R, T)$ , the ISA agent first updates its network reaction time estimator  $T_s$ , by using  $T$  as a new sample input to the average smoothing function described before. If the agent is a current  $R_h$  receiver, it immediately drops the highest layer. Otherwise, if the agent detects that it now belongs to the set of agents subscribing to the new highest level, it updates its timers and mode as if this drop action was taken by itself. This is important to minimizing probing overhead within a domain. Since the whole domain has unsubscribed to the highest layer. This implies that all agents in the domain should go to the S (sleep) state immediately to avoid further unnecessary reactions to overloads that were potentially caused by the highest layer. The following pseudo code describes the ISA agent's behavior in response to receiving a  $\text{DROP\_ACK}(R, T)$  message.

```

if ( ACK timer is scheduled )
    cancel timer ;
 $T_s = \alpha_s T_s + (1 - \alpha_s) T$  ;
 $R_h = R$  ;
if (  $R_c > R$  ) {
     $R_c = R$  ;
    drop layer(s) ;
}else if (  $R_c == R$  )
    update mode and timers as if this drop happened locally ;
goto state S ;

```

### Handling DROP\_NAK messages

On the contrary to  $\text{DROP\_ACK}$  messages which are of interest to all agents in the domain to learn the new  $R_h$  level,  $\text{DROP\_NAK}$  messages are of interest only to the agent who sent a  $\text{DROP\_REQ}$  in the first place. This interest holds true as long as that sender neither received a  $\text{DROP\_ACK}$  nor did it send one. The following code

is executed by an agent on receiving a DROP\_NAK message.

```
if ( ACK timer is scheduled )
```

```
    NAK_Received = True ;
```

### Handling ACK timer expiration events

Expiration of the ACK timer at the agent which originally sent the DROP\_REQ message, means that no DROP\_ACK messages were received. This implies that either only DROP\_NAK replies were received or no replies were received at all. In the former case, it is concluded that the congestion is local to the host which detected it and no further action is taken. In the latter case, the agent concludes that it is the only agent subscribing to  $R_h$  and therefore having degraded its level of subscription implies that the new  $R_h$  should be announced via a DROP\_ACK message to the whole domain.

```
if (NAK_Received)
```

```
    return ;
```

```
else {
```

```
     $R_h = R_c$  ;
```

```
    send DROP_ACK( $R_h$ ,  $T_s$ ) ;
```

```
}
```

### Handling congestion

Although there is a high likelihood that congestion developing in a domain can be solved by reducing the total rate received from all streams, or in other words by the agents subscribing to the highest layers to reduce their subscriptions and hence reduce the network load, yet there are situations where local overload in a host will develop. In such situations the ISA agent on that host should take degradation decisions even though it is not subscribing to the highest layer. However, if we let the agent react immediately (as soon as its  $T_u$  timer expires), we may render the whole shared learning process useless in the former cases of network load, since

its reaction time will be almost equivalent to the highest layer subscribers reaction time. The solution to this problem is to ensure that the low rate receivers have longer reaction times than the high rate receivers. Thus if a network load situation develops the high rate receivers react first, while if a host load situation develops at a low rate receiver reaction is still guaranteed.

To achieve this goal, an agent moving into the U (Underqualified) state checks first its  $R_c$  and if it is less than  $R_h$  it extends the  $T_u$  timer to  $T_u^{max} + 1$ . This ensures that any network congestion is detected first by highest rate receivers in the domain, since for those receivers  $T_u \leq T_u^{max}$  always. This is done only if the agent is either in the *Enhance* or the *Probe* mode. However, if the agent is in the *Degrade* mode, i.e., it had already dropped layers in spite of not subscribing to the highest layer in the domain, its  $T_u$  timer is not extended and is updated according to the mode as described in previous sections, to avoid unnecessary additional delays in reacting to the host load.

### Protocol robustness

As previously mentioned, the domain rate control protocol is a supplementary enhancement for the scalability of the system by reducing the total number of reactions made in a domain. When deployed with the recommended TTL of one, loss of any of the protocol messages is not expected except in the unlikely event of subnet sustained overload for a relatively long period of time. However, even if such a situation develops, the worst case that could happen to an agent is to lose one or more of the protocol messages leading to wrong information stored in the state variable  $R_h$ . We show here that this situation will be quickly corrected, and will not lead to any serious conditions. Let  $R'_h$  be the wrong value at an agent (hereafter referred to as “faulty agent”) whose current operating rate is  $R_c$ , while  $R_h$  is the correct value at all other agents. One of the following scenarios may develop. It should be noted that  $R_c$  cannot exceed  $R'_h$  for the faulty agent.



1.  $R_c < R'_h$ . In this case, the faulty agent will not have any active role in the protocol and hence will not confuse the other agents. If  $R_c < R_h$  as well, the next ADD or DROP\_ACK message in the domain will synchronize the value of  $R'_H$  to the rest of the domain. On the other hand, if  $R_c > R_h$ , the faulty agent will ignore all messages from the the other agents (low rate agents with respect to the faulty agent), except for DROP\_ACK messages which synchronize the values of  $R_h$  in the whole domain, including the faulty agent. This synchronization will force the faulty agent to drop the layer(s) above the  $R_h$  decided by other agents in the domain.
2.  $R_c = R'_h$ . In this case, the faulty agent will become active and will send protocol messages if it decides to add or drop a layer. If  $R'_h < R_h$ , ADD messages sent by the faulty agent will be ignored by the other agents, but a DROP\_ACK will synchronize the values of  $R_h$  in the whole domain to the new level decided by the faulty agent. This is an undesired behavior, however, its likelihood is minimal since having  $R'_h < R_h$  means that one or more ADD messages were lost which is unlikely because ADD messages are sent only when the domain conditions are favorable. On the other hand, if  $R'_h > R_h$ , then if the faulty agent adds or drops a layer the  $R_h$  values in the whole domain will be synchronized to  $R'_h$ .

In spite of the above evidence of protocol robustness, one possible simple solution to increase the reliability of the protocol messages, is to send each message more than once, with enough provisions to identify redundant messages from the same source at the receiver. Alternatively, a simple LAN oriented reliable multicast protocol may be suitable for this purpose.

### Summary of advantages of domain rate control

- **Shared learning of network reaction time.** This is achieved by  $T_i$  announcements at the end of a failed join experiment, thus allowing those re-

ceivers which did not participate in the experiment to learn from it.

- **Stability of low-end receivers.** Since high-end receivers are made to react first when congestion is detected in a domain, low-end receivers do not react to domain congestion caused by higher layers which are brought into the domain by the high-end receivers.
- **Minimizing the number of independent probes in the domain.** This is achieved by synchronizing the probing action for receivers which probe for a rate above the current stable rate in the domain. This in turn minimizes the potential disturbances caused by probing.
- **Enhancing responsiveness in reaction to congestion.** This is achieved by synchronizing the action of dropping the highest layer in the domain. As soon as network congestion is detected by at least two highest layer receivers, the other receivers receiving this layer in the domain are forced to drop it at once, without waiting for them to detect the congestion at their own pace.
- **Robustness.** The domain rate control protocol is merely an optimization for the performance of the QoSess layer. If some of the protocol messages are lost, the QoSess layer still works albeit with potentially reduced performance.

## 5.4 Evaluation

The concept of shared learning, where receivers of a multicast stream learn from the *join experiments* of other receivers, was first introduced in the RLM protocol [46]. In RLM, a fully distributed approach to rate control is taken. A receiver performing a join experiment makes an announcement to the whole group. It is then up to each receiver to make its own conclusions and learn from this experiment.

This approach has several drawbacks. First of all, the load introduced by making announcements to very far receivers that couldn't possibly benefit from

the experiment is unjustified. Additionally, these announcements from far away receivers may confuse others if they correlate overloads coincidentally developing in their domains to the active join experiment in a different domain. Furthermore, even within the same domain, confusion may happen because the receiver whose join experiment failed does not explicitly announce that. Thus, others may correlate an overload to the active join experiment, while the fact that the joiner did not suffer from that overload clearly implies that this overload is due to other conditions developing in the network path or host which detected it and needs its reaction.

In domain rate control, we avoid these drawbacks by grouping receivers which are definitely affected by the same network conditions into separate domains, and by explicit announcements, scoped within the domain, of not only the start of join experiments (ADD messages) but their failure (DROP\_ACK messages) as well.

In the LVMR protocol, rate control by shared learning among receivers of a multicast stream is achieved through installing a set of managers arranged in a hierarchy that matches the structure of the multicast tree rooted at the sender [44]. In this way, correct correlations between join experiments and congestion resulting from these experiments can be made across several subnets. However, the hierarchical structure fits naturally a sender and a group of receivers for one stream. It does not fit the nature of a distributed session in which multiple senders for multiple cooperating streams co-exist. One of the drawbacks of this approach is the need for knowing the structure of the multicast routing tree constructed by the routers in order to install the intermediate managers appropriately. Such an interaction between the protocol and the routing protocol is not defined. At the least, knowledge about the wide area network topology is necessary for the proper arrangement of the managers. Also, although more than one timer are backed-off and relaxed based on the current network conditions, as in RLM and in QoSess, yet LVMR, in contrary to the other two protocols, lacks a component for learning the long term network reaction time.

A major difference between the QoSess approach and the other two ap-

TABLE 5.4  
COMPARISON OF RECEIVER DRIVEN RATE CONTROL PROTOCOLS

	RLM	LVMR	QoSSess
Number of streams	one	one	many
Timers nature	layer-specific	layer-specific	mode-specific
QoS measure	loss	loss, late arrival	loss, jitter
Shared learning	fully distributed	hierarchical	domain
Learn network delay	yes (conservative)	no	yes (aggressive)
Collaborative layer drop	no	yes	yes
Parallel probing	may cause confusion	all layers synchronized	highest layer synchronized
Use capacity info	no	no	yes (if available)
Retransmission	no	yes	no

proaches, besides handling multiple streams, is the usage of multi-modal timers, as explained before. Both RLM and LVMR use layer-specific timers as opposed to mode-specific timers. A potential problem with layer-specific timers is the excessive prolonging of timers for joining some of the layers during overload conditions that made the system stabilize just below such layers at different times during the session. Later on, when conditions are favorable, enhancement may be delayed for long times at arbitrary layers. The same problem may arise during degradation when a sudden congestion condition faces very slow reactions at arbitrary layers due to previous prolonging of these layers' timers. Table 5.4 summarizes the differences between the three protocols.

## 5.5 Experimental Results

In this section, we demonstrate the stability and responsiveness of the QoSess layer, through a set of experiments conducted using a prototype implementation of the QoSess control layer. In these experiments, the following empirically derived values were used for the different parameters (all time parameters are in seconds), unless otherwise explicitly specified:  $T_o^{min} = 1$ ,  $T_o^{max} = 8$ ,  $T_u^{min} = 2$ ,  $T_u^{max} = 4$ ,  $T_p^{min} = 8$ ,  $T_p^{max} = 60$ ,  $T_s^{min} = 2$ ,  $T_s^{max} = 16$ ,  $T_u^{join} = 1$ , and  $\alpha_o = \alpha_u = \alpha_p = 2$ .

### Baseline stability

First, we examine the baseline stability offered by the ISA state transition diagram using the multi-modal timers described in Section 5.3. For that purpose, we contrasted the performance of the ISA agent in three different cases:

1. Constant  $T_o$  and  $T_u$  timeout parameters, of 2 seconds each, were used for the hysteresis O and U states.
2. The constant timeouts solution was augmented with smoothing of the measured losses, in order to achieve better stability in the ISA state transitions. Losses were smoothed as follows:  $Loss = \alpha_L Loss + (1 - \alpha_L) Current\_Loss$ .
3. Multi-modal timers were used as explained in Section 5.3, with immediate loss notification, i.e., without smoothing the measured losses.

In this first set of experiments, only one stream was used. The stream was produced by a hierarchical video encoder which produces three constant bit rate layers of 45, 180, and 525 Kbps for 15 frames per sec video [52]. After 30 seconds from the beginning of the experiment, uniform losses were induced constantly for another 30 seconds. Two cases were examined corresponding to 15% and 30% loss ratios.

In Figure 5.6, constant hysteresis timeouts were used.  $T_o$  and  $T_u$  were both set to 2 seconds. As shown in the figure, although congestion was detected almost

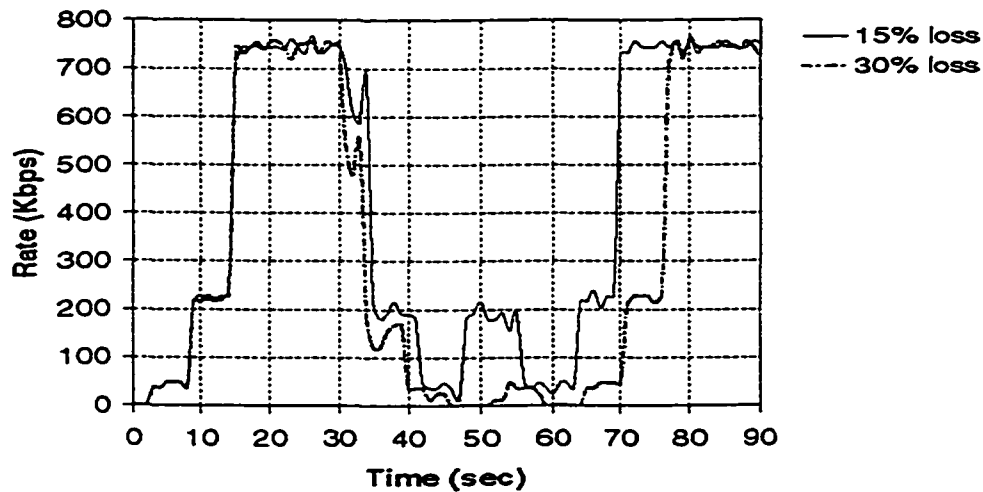


Fig. 5.6. Constant timers and stability.

instantaneously because the losses were induced by a loss module augmented to the receiver, yet the system oscillated in spite of the sustained losses. These oscillations occurred due to false detection of favorable conditions when a few consecutive packets were not lost.

Figure 5.7 shows that smoothing the measured losses did not prevent oscillations, while causing an increased delay in reaction to congestion, especially in the 15% losses case, which took longer time to produce a congestion indication. This was true for values of  $\alpha_L$  ranging from 0.25 to 0.75.

Figure 5.8 depicts the performance when multi-modal timers were used. As can be seen from the figure, no oscillations occurred in this case, while the time taken to react to congestion lied between the two previous cases. However, a slight increase was observed in the time taken to enhance the perceived quality of session, by adding more layers after the congestion was over.

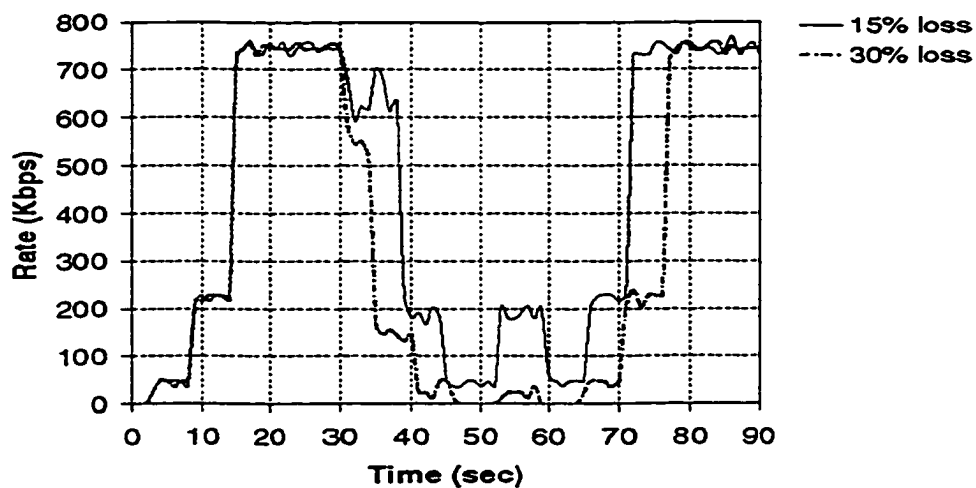


Fig. 5.7. Smoothed losses and stability.

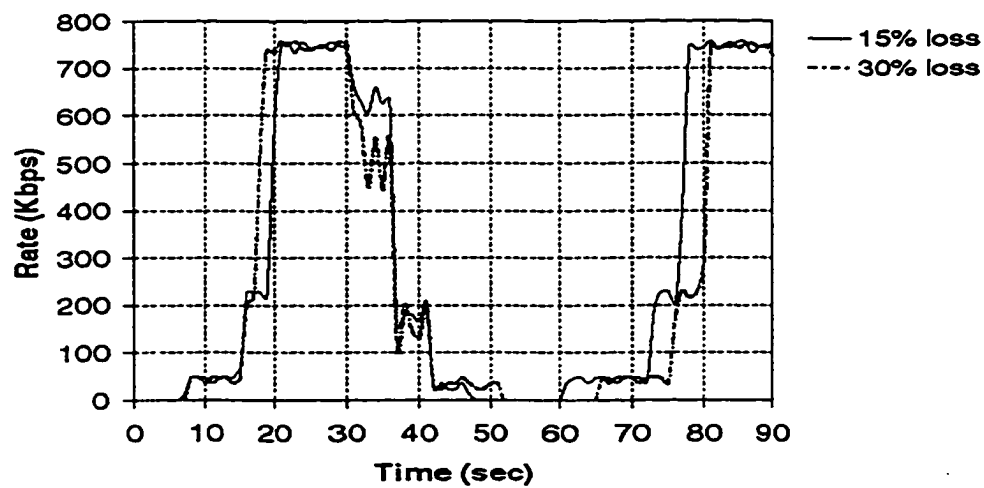


Fig. 5.8. Multi-modal timers and stability.

## Reacting to subnet overload

The following experiment was conducted in order to illustrate several aspects of the QoSess layer. These aspects are listed below.

1. The ability of the QoSess layer to react to subnet overload conditions resulting from locally increased offered load to the subnet.
2. The importance of accounting for congestion signals other than losses. Specifically, the jitter effectiveness in detecting subnet overload conditions is illustrated.
3. The importance of the feedback protocol in eliminating undesired traffic in Intranet collaboration environments.

In this experiment, a distance learning session is composed of three video streams: a higher priority teacher video stream (TV), and two lower priority student video streams (SV1 and SV2). Each of the streams is hierarchically encoded by an encoder which produces three constant bit rate layers of 45, 180, and 525 Kbps for 15 frames per second video [52]. *G2P* mapping yielded 2 for the priority of TV and 1 for the priority of each of the other two streams. The session was conducted over an Intranet composed of 10 Mbps switched Ethernets. Therefore, router-based protocols, like IGMP [24], for eliminating undesired layers did not exist. The session depended solely on the QoSess layer feedback protocol to eliminate undesired layers.

After 60 seconds from the beginning of the session, the Ethernet was overloaded by two workstations, which do not belong to the session, exchanging uncontrolled high rate traffic targeting 60% of the Ethernet capacity, for 1 minute. As well known, Ethernet performance starts to deteriorate sharply at 60% load, due to medium access collisions.

Figure 5.9 contrasts the aggregate received rate, as measured by a session member, in presence and absence of the QoSess control layer. It is clear from the figure that the QoSess layer was able to capture and control the overload situation.



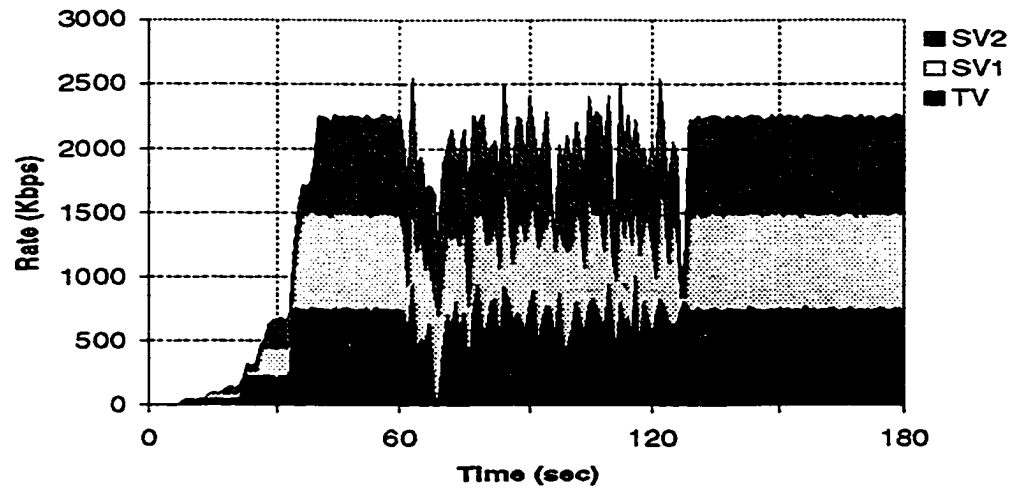
As Figures 5.10 and 5.11 indicate, the QoSess layer controlled both the losses and jitter caused by this overload condition. This was achieved not only by degrading the operating rates of the receivers, but by reducing the transmission rates of the senders as well. This control over the senders transmission rates was achieved via the feedback protocol described in Chapter 4. This led to decreasing the overall load offered to the Ethernet, thus allowing the session to proceed at a degraded, but smooth (no losses or jitter,) quality.

This control would have been much difficult, if at all achievable, if the QoSess layer depended only on losses as the sole congestion signal, or indicator for the actual offered QoS. Due to the excessive buffering in the end-hosts (compared to intermediate routers,) an increase in the offered load in a subnet environment, and particularly in the Ethernet case, would lead to losses only if that load is too high or if sustained for relatively long periods. Depending on jitter as a measure for the actual QoS, besides losses, is what allowed for achieving this smooth operating level, despite the high sustained load offered to the Ethernet.

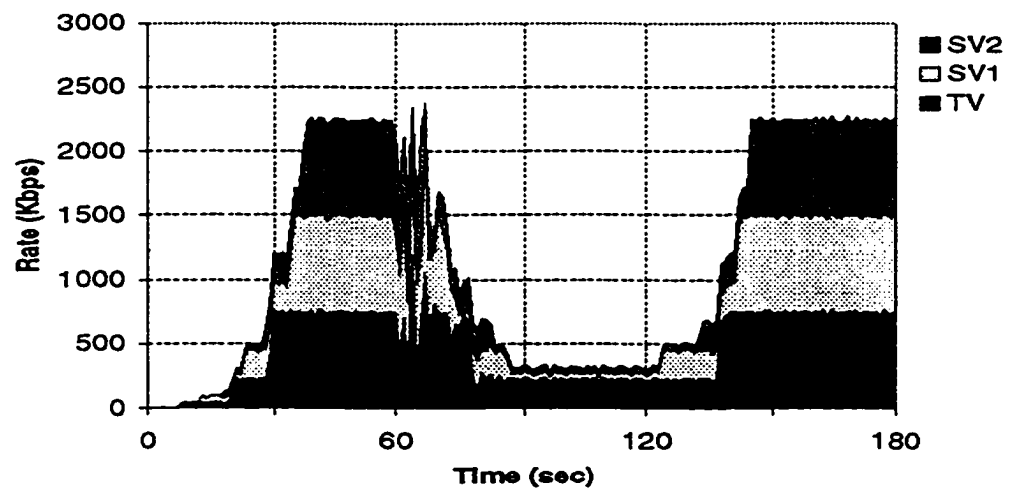
As Figure 5.11(b) shows, residual jitter existed throughout the duration of the cross traffic. This was an indication to the QoSess layer that the load condition was still in place. If this signal was ignored, the layer would have detected favorable conditions, decided to enhance the quality of the session by joining more layers, which would cause eventual losses that imply re-degradation actions. Thus, the system would oscillate, if jitter was not accounted for as a QoS indicator.

### **Reacting to network congestion**

Figure 5.12 illustrates the adaptability of the ISA agent to available bandwidth in an inter-networking setup. In this experiment, a distance learning session was composed of three video streams as described before. The figure plots the cumulative rate received by a receiver sitting behind a bottleneck link with 1.5 Mbps transmission rate and 15 KB (15 packets) router buffer size, which is well above double the bandwidth-delay product of the link. For the purpose of this experiment and

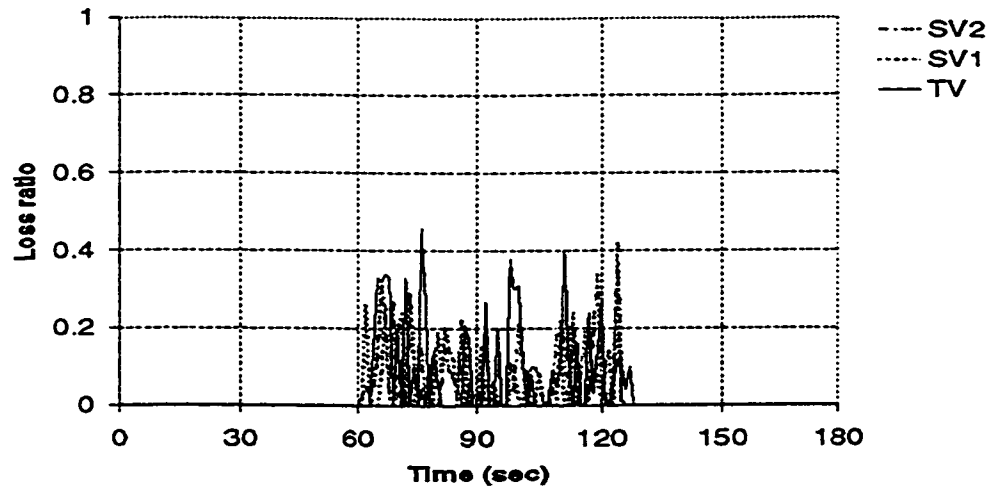


(a)

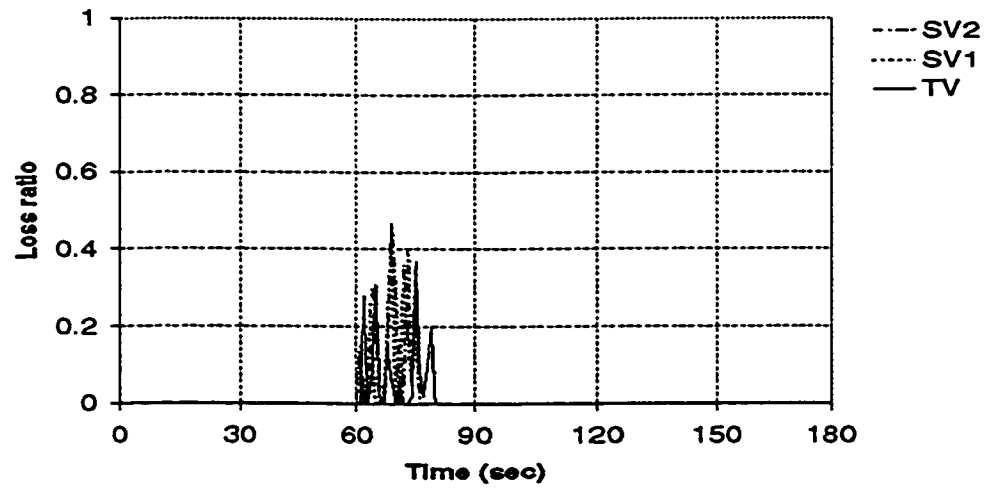


(b)

Fig. 5.9. Aggregate throughput at a receiver on a congested LAN. (a) Without QoS. (b) With QoS.

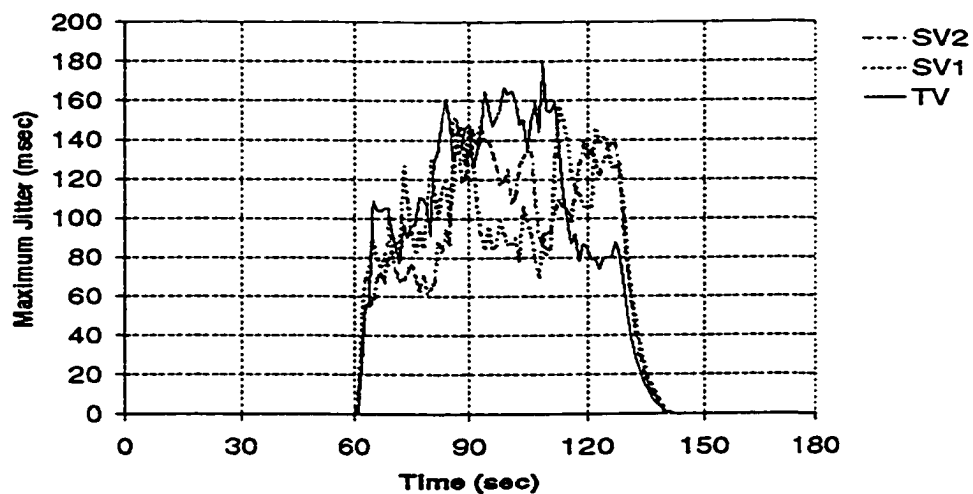


(a)

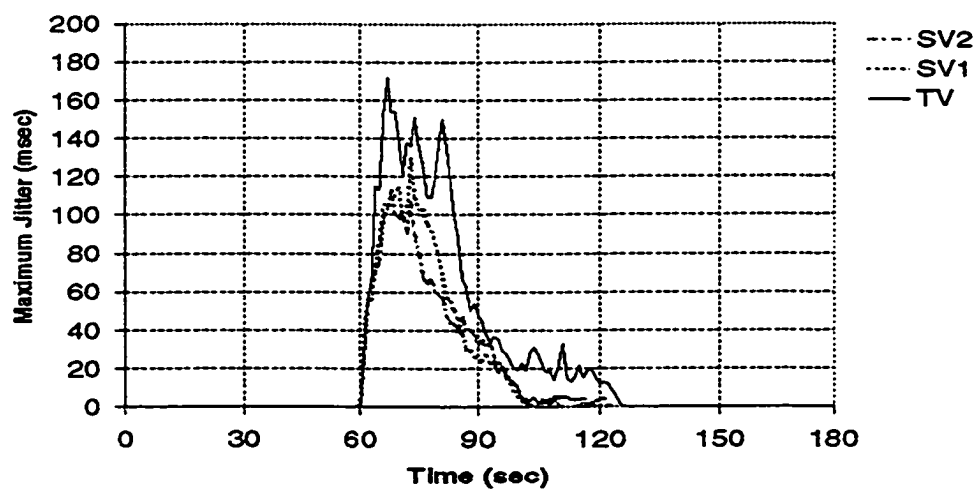


(b)

Fig. 5.10. Losses at a receiver on a congested LAN. (a) Without QoS. (b) With QoS.



(a)



(b)

Fig. 5.11. Jitter at a receiver on a congested LAN. (a) Without QoS. (b) With QoS.

others to be illustrated below, we developed our own configurable router emulation software.

As shown in Figure 5.12, the ISA agent switches to the *Enhance* mode soon after the session starts and quickly reaches the stable subscription level. Then the agent switches to the *Probe* mode where the time between two consecutive probes gets backed off over time. After 180 sec from the beginning of the session, the bottleneck link is subjected to a cross traffic load of 1.2 Mbps for 1 minute. As soon as the congestion is detected, the agent switches to the *Degrade* mode where drop hysteresis is reduced and thus it quickly drops layers and stabilizes at a low rate, where it switches to the *Probe* mode and then to the *Enhance* and finally *Probe* mode again.

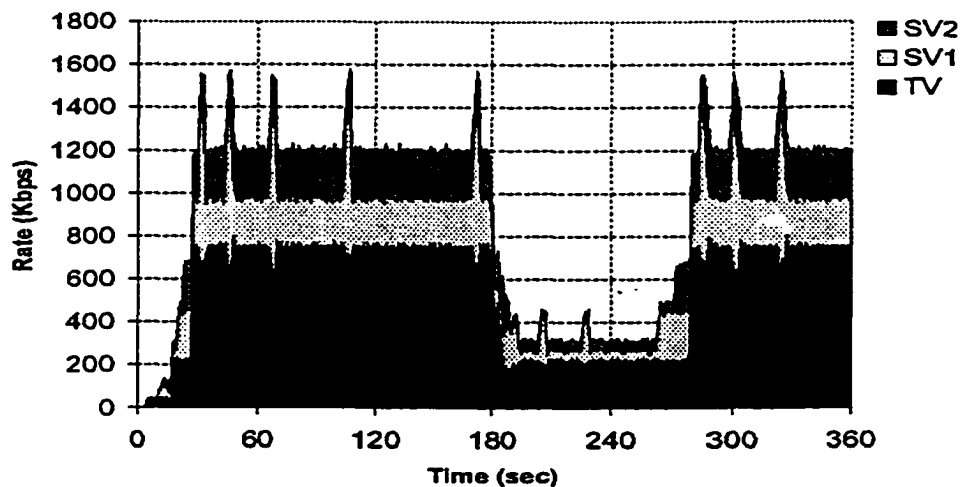


Fig. 5.12. Aggregate throughput at a receiver behind a bottleneck link.

### Learning network delays caused by large buffers

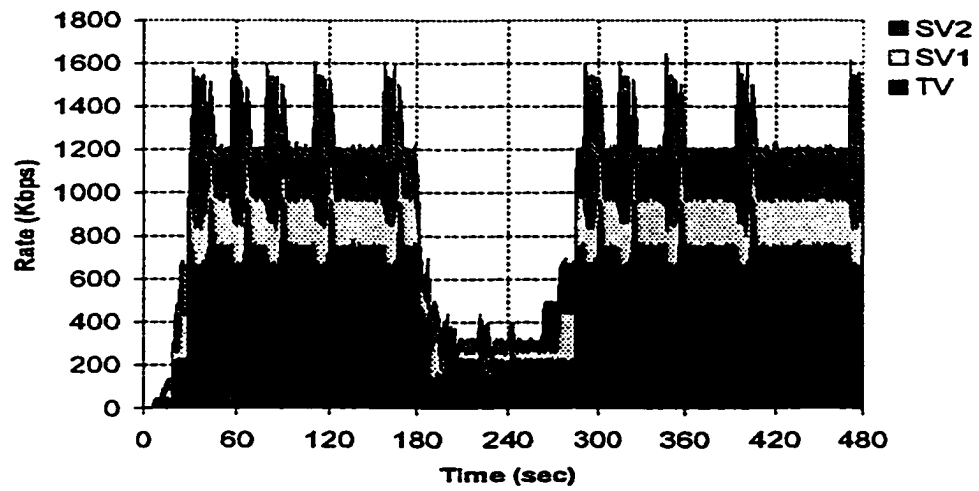
Figure 5.13(a) shows the result of performing the same experiment as above, when the buffer size at the bottleneck link router was multiplied 10 times its original value, i.e., set to 150 KB. This excessive buffering leads to a delay between the time a congestion starts to develop and the time of its detection. Similarly, when an action is taken to alleviate congestion, it takes the network a delay period before the effect of the action is pronounced. This is because of the packets already generated and queued under previous conditions. Nevertheless, as the figure shows, the ISA agent manages to capture this delay characteristic of the network and performs in a way similar to Figure 5.12. The main difference between the two cases is in the time-spacing of the first few probes at the beginning of the session. In the larger buffer case, it takes the agent a while to adapt the sleep timeout,  $T_s$ , to the network delay. As shown in Figure 5.13(b), this adaptation takes about 90 seconds, since  $T_s$  is initially set to 2 seconds, which is a relatively small value.

As the router buffers become occupied with more cross traffic than the session traffic, probing leads to quicker losses occurring in the session streams. This is demonstrated by the temporary decrease in the value of  $T_s$  during the congestion period.

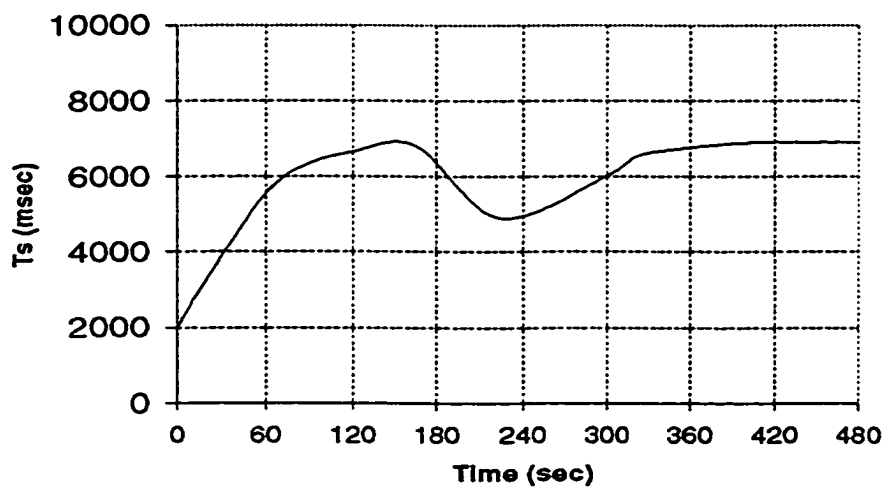
### Learning network delays caused by slow links

Figure 5.14 shows the output of a session composed of three video streams as above. In this experiment, instead of increasing the buffering space, the delay of the bottleneck link was set to 5 seconds. This is a relatively large delay; larger than typical satellite total up-link plus down-link delays.

In spite of the presence of such excessive link delay, and in spite of the relatively small  $T_s$  initial value which is not suitable for this environment, the ISA agent was able to learn this network delay and adapt  $T_s$  to it reasonably within two minutes from the beginning of the session. However, it should be noted that a more



(a)



(b)

Fig. 5.13. Adapting the sleep timeout ( $T_s$ ) to large buffers. (a) Aggregate throughput. (b) Sleep timeout ( $T_s$ ).

conservative initial setup for  $T$ , leads to faster learning of the network delay, in such circumstances. An initial value of as low as 4 seconds was sufficient to avoid the oscillation that happened near the end of the first minute of this experiment.

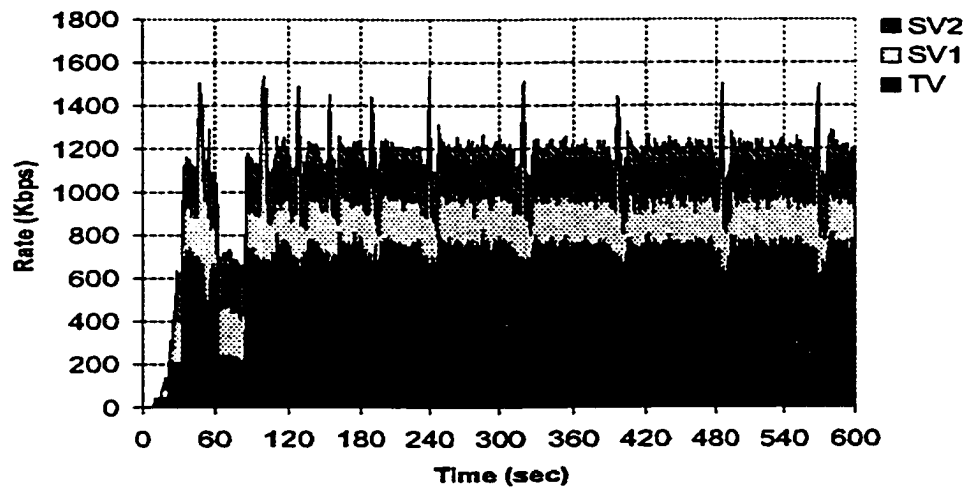
## 5.6 Conclusions

In this chapter, we devised an architecture for realizing the quality of session control framework. The architecture is composed of two types of agents: monitoring agents and inter-stream adaptation (ISA) agents. The software design of the agents was illustrated, and the principles guiding the design of the QoSess control layer were discussed.

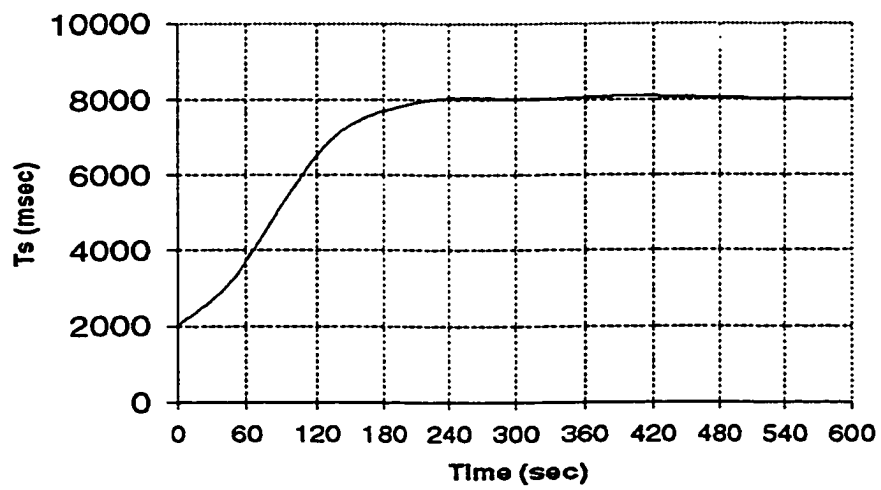
In order to support heterogeneity of receivers and networks, a key design decision was to adopt a receiver-driven approach, where each participating host decides for itself which layers of which streams to receive. This receiver-driven approach is the key for scalability. Additionally, it allows each ISA agent to employ techniques such as multi-modal timers and learning network reaction time from its own perspective to achieve stability at the controlled host. Moreover, the co-existence of several ISA agents in the same session opens further avenues for cooperation among those agents to enhance the stability of the system. To this end, we introduced a domain rate control protocol. In this protocol, neighboring ISA agents cooperate together and coordinate their actions in order to manage the session rate in their domain. This coordination is done by sharing knowledge, through minimal exchange of control messages, in a way that prevents congestion while minimizing the amount of work needed to be done by individual agents to estimate the available capacity in the domain.

A prototype QoSess layer was implemented based on the proposed architecture, and the mechanisms suggested in this chapter and the previous two chapters. Experiments conducted using the prototype system verified the stability and responsiveness of the QoSess control layer. These experiments demonstrated the ability of





(a)



(b)

Fig. 5.14. Adapting the sleep timeout ( $T_s$ ) to large delays. (a) Aggregate throughput. (b) Sleep timeout ( $T_s$ ).

the layer to react timely and appropriately to overload and congestion conditions, while satisfying the application-specific dynamic constraints regarding the relative importance of the different streams to the session.

## CHAPTER VI

# CONCLUSION AND FUTURE EXTENSIONS

### 6.1 Conclusion

Inexpensive hardware for processing digitized audio and video data is rapidly becoming available for workstations and desktop computers. At the same time, high network bandwidth at relatively low price is widely available at the desktop. These developments have stimulated interest in the application of computer-based conferencing (using audio and video as well as data) to support effective collaboration among teams of workers in various fields [1, 45]. Thus, continuous media streams represent a major component of new distributed collaborative systems. These continuous media streams have some inherent characteristics that are not found in other data streams; they have timing and throughput requirements that must be met.

Several researchers proposed different proactive and reactive solutions to support the requirements of multimedia streams. However, these solutions typically manage the Quality of Service (QoS) offered to individual connections in isolation of others. The inefficiency of this model, of independent management of streams, in handling the requirements of interactive multimedia collaborative (IMC) applications was realized recently, and the proactive and reactive approaches were both ratified. In the proactive approach, group reservation schemes were proposed whereby collective resources can be reserved and shared by several streams belonging to an application [35, 67]. In the reactive approach, differential service models

were proposed to enhance the traditional best-effort service model, by providing distinct service to groups of streams classified as belonging to certain sessions, or in general, as belonging to an identifiable class of packets [28]. In spite of the aforementioned recognition and progress towards aggregate management of streams belonging to an individual class or session, current solutions and trends in this direction do not satisfy all the fundamental requirements of IMC applications. These solutions provide basic network-level mechanisms for supporting aggregate management of connections. However, the notion of cooperation among a set of streams to achieve a unified goal is not considered, and hence, higher level resource allocation policies and mechanisms are required. These mechanisms must adapt the application streams in order to avoid congestion, while supporting the cooperative nature and dynamic behavior of the streams, and accounting for the heterogeneity in network and receiver capabilities.

For these reasons, we opted to devise an application-oriented framework for achieving global control over the quality of IMC sessions. The *Quality of Session (QoSess)* framework, presented in this dissertation, controls the QoS offered by the system across the set of connections belonging to the IMC application, in order to avoid any potential competition for resources among the streams of a session. This control is based on the application semantics, with the objective of maintaining the best overall quality of session, throughout the session lifetime. The framework does not require the network to provide group management schemes, although the presence of such schemes as group reservations or differential services is bound to enhance the performance. In this framework, a QoSess layer constantly monitors the observed network behavior with respect to the session streams, makes inter-stream adaptation decisions, and sets the new operating level for each stream from within its permissible range of operating points.

The problem of inter-stream bandwidth adaptation was thoroughly investigated in this work. The relative importance of the different streams to the session was represented by a QoSess graph. The QoSess graph enabled the separation of

inter-stream adaptation policies from mechanisms. Two inter-stream adaptation algorithms, RISA and I-WFS, were devised. The behavior of the two algorithms was studied using two types of traffic: constant bit rate traffic; and traffic characterized using the M-LBAP model. Simulation results showed that significant enhancements in resource utilization and acceptance ratios are always achievable using these inter-stream adaptation mechanisms, relative to the traditional static resource allocation policies, for groups of streams which are cooperating to fulfill a unified global goal, and each is willing to sacrifice for the sake of the benefit of the whole group. In the absence of group reservation support in the network, the inter-stream bandwidth adaptation mechanism used should be able to operate correctly without knowledge about the bandwidth available to the session. Additionally, multimedia sources are typically able to vary their transmission rates in discrete steps only. The RISA algorithm was shown to support these two constraints, and a new inter-stream bandwidth adaptation algorithm, A-IWFS, was devised to approximate the behavior of I-WFS under these additional constraints. The performance of A-IWFS was studied, and it was shown that adhering to this inter-stream adaptation algorithm, when reacting to dynamic congestion conditions or static capacity constraints, leads to efficiency and fairness in bandwidth utilization, yielding higher session fidelity.

A problem that often arises in multicast systems in general, and in multimedia multicast systems specifically, is the need for soliciting feedback information from a group of receivers, in a scalable manner. The QoSess framework presented in this dissertation is no exception to that. Providing the source of a stream with feedback information about the used layers of the stream is crucial for the efficient utilization of the available resources. The feedback mechanism allows the sender to always encode and send only layers for which interested receivers exist, and to suppress unused layers. Deploying such a feedback mechanism is not merely an added efficiency feature, but it is a critical feature for the success of IMC sessions in which multiple streams are concurrently active. In presence of scarce resources, inter-stream adaptation decisions sacrifice the quality of lower priority streams for

the sake of releasing resources to be used by higher priority streams. If the low priority sources keep pushing all unused layers to the network, the decision made by the receivers to drop these layers for releasing resources is rendered useless. This uselessness will hold true forever for the source host and LAN, while the rest of the network may eventually have these resources released as the multicast routers stop forwarding the unused layers. Besides the unnecessary delay in releasing resources, the fact that the source host and LAN will always be overloaded is very critical, as the session participants on this LAN may not be able to receive other higher priority streams. The problem is more critical for Intranet based collaboration systems since all the session participants (senders and receivers) may be within a few hops from one another (see for example [45]), and the subnets may be interconnected via non-intelligent switches that are incapable of dropping unused layers.

For these reasons, we devised and simulated a new scalable and robust state feedback protocol. The protocol is used for controlling the source transmission rate in both cases of layered and single-rate multicast. It allows for determining the worst case state among a group of receivers, where each receiver may be in one of a set of finite states, and is applicable in receiver-driven as well as in sender-driven adaptive multimedia systems. Simulation results showed that the presented feedback protocol scales well for very large groups of up to few thousands of participants. The efficiency of the proposed protocol in eliminating the reply implosion problem, its robustness in facing network losses, as well as its responsiveness were illustrated. For typical sessions with up to 100 participants (e.g., IRI sessions [45]), less than 10% of the receivers reply to a probe, in worst case topologies, while for larger sessions, of a few thousands of participants, the reply ratio is below 1.5%. The average response time was found to be always below the maximum round-trip time from the sender to the group members. In addition, the advantages of the proposed protocol over other protocols, which commonly rely on session level messages for estimating individual round-trip times from each receiver to the sender, were demonstrated. Moreover, adaptive enhancements were proposed to maintain the protocol scalability for even

larger groups of up to 10,000 participants.

An architecture for realizing the proposed framework, and incorporating the devised inter-stream adaptation and feedback techniques, was proposed and prototyped. The architecture is composed of *monitoring agents* and *inter-stream adaptation (ISA) agents* that implement together the policies and mechanisms needed for QoSess control. In order to scale in heterogeneous environments, a receiver-driven approach was adopted, where decisions are made locally at each host regarding which layers to receive. Additionally, several provisions, including a domain rate control protocol, were devised in order to ensure stability and responsiveness in the inter-stream adaptation process. Experiments conducted using the prototype system confirmed that the QoSess control framework enhances the attainable overall quality of session in both favorable and congested network conditions. These enhancements manifest themselves as efficient utilization of the instantaneous available resources, and the allocation of these resources among the session streams in a way that prevents competition and satisfies the application dynamically dictated relative priority constraints.

## 6.2 Future Extensions

The work presented in this dissertation may be extended in several ways as described below.

- Extending the two presented inter-stream adaptation algorithms, RISA and I-WFS, to utilize the QoSess graph directly as input without the need to map the graph into priority classes. Although this introduces an extra level of complexity to the algorithm, it is expected to yield more efficient utilization of the available resources.
- Devising new inter-stream adaptation policies and algorithms.

- Standardizing the QoSess framework. In this dissertation, we did not only present a framework for achieving quality of session control, but we presented and prototyped a possible architecture for realizing this framework as well. Adapting this architecture in order to be integrated with RTP [51], the only standard media transmission protocol, seems an appropriate step towards standardization of the QoSess framework. This process involves producing IETF drafts for ratification of RTP.
- Enhancing the scalability of RTCP, the feedback control protocol associated with RTP, using the scalable state feedback protocol presented in this dissertation. This step also involves producing ratification drafts to the IETF.
- Experimentally investigating the gain from deploying the QoSess framework in best-effort networks whose service model is enhanced by supporting differential services. Successful nation-wide communication over differential services was recently demonstrated [28]. In contrast to reservation solutions based on RSVP [67], the deployment of differential services in the Internet on a large scale is anticipated shortly. The need and potential benefit from deploying QoSess control mechanisms in this environment is evident. However experimental demonstration and quantification of such benefits is lacking.
- In general, investigating the effect and potential interaction between the QoSess layer performance and different router queuing schemes, other than the commonly used FIFO drop-tail queues. Examples for other promising queuing schemes, which are currently under investigation for deployment in the Internet, include class-based queuing (CBQ) [33] and randomized early drop (RED) queues [11, 32].
- The network delay estimation technique used in setting the sleep timeout of the ISA agent assumes symmetric network delays in reaction to both add and drop operations. Some recent preliminary reports [44] indicate that this may



not be true. However, the congestion signals used in producing these reports were not as sensitive as the jitter signal that we account for. Nevertheless, a careful investigation of this issue seems useful. Should the results presented in these preliminary reports be confirmed in spite of using more sensitive congestion measures, asymmetric sleep timeouts may be used for the add and drop operations.

- Incorporating more complex application semantic requirements and relationships among streams. A simple language for representing these requirements could be devised, or alternatively, new evolving standard languages for representing media play-out specifications, such as SMIL (Synchronized Multimedia Integration Language) [58], could be adapted or extended for this purpose.
- Extending the proposed inter-stream adaptation mechanisms, in order to handle independent path streams. Some leaf subnets may be connected to the outside world by means of more than one router, thus, more than one independent path may lead to a receiver. In this case, different streams may take different paths to a receiver. Therefore, it is desired to adapt only the performance of the streams that share a bottleneck connection without affecting streams which are using a separate path. The same problem may also arise for a receiver whose subnet is connected by one router only. Typically, it is expected that congestion develops at links closer to a receiver, because these links are subjected to the load of all the session streams, especially with center-based multicast routing [25]. However, the probability of congestion happening at bottleneck links close to the senders still exists. In such situations, it is desired that the adaptation mechanism, deployed by the ISA agent, adapts only the performance of the affected streams. Information regarding which streams are affected by a congestion occurring in the network is readily available to the ISA agents through the reports obtained from the monitoring agents. What is needed is extensions to the adaptation mechanisms in order

to accommodate more than one simultaneous probing and capacity estimation activity, one for each independent path, per receiver. This is expected to yield better utilization of available resources.

- Finally, investigating other possible architectures, besides the one proposed in this dissertation, for realizing the QoSess control framework.

## REFERENCES

- [1] H. Abdel-Wahab and M.A. Feit, "XTV: a framework for sharing X window clients in remote synchronous collaboration," *Proc. IEEE Conference on Communications Software: Communications for Distributed Applications and Systems*, pp. 159-167, Chapel Hill, NC, April 1991.
- [2] H. Abdel-Wahab, K. Maly, A. Youssef, E. Stoica, C.M. Overstreet, C. Wild, and A. Gupta, "The Software Architecture and Interprocess Communications of IRI: an Internet-based Interactive Distance Learning System," *Proc. IEEE Fifth Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'96)*, pp. 4-9, Stanford, California, June 1996.
- [3] E. Amir, S. McCane, and H. Zhang, "An Application Level Video Gateway," *Proc. ACM Multimedia*, San Francisco, CA, November 1995.
- [4] E. Amir, S. McCane, and R. Katz, "Receiver-driven Bandwidth Adaptation for Light-Weight Sessions," *Proc. ACM Multimedia*, pp. 415-426, Seattle, WA, November 1997.
- [5] D.P. Anderson, "Meta-scheduling for distributed continuous media," *ACM Trans. Computer Systems*, vol. 11, no. 3, August 1993.
- [6] A. Banerjea, D. Ferrari, et al., "The Tenet Real-Time Protocol Suite: Design, Implementation, and Experiences," Technical Report TR-94-059, International Computer Science Institute, Berkeley, California, November 1994.
- [7] R. Bettati, D. Ferrari, A. Gupta, et al., "Connection Establishment for Multi-Party Real-Time Communication," *Proc. Fifth International Workshop on Network and Operating System Support for Digital Audio and Video*, Durham, New

- Hampshire, April 1995.
- [8] R. Bolla, M. Marchese, and S. Zappatore, "A Congestion Control Scheme for Multimedia Traffic in Packet Switching Best-Effort Networks," *Proc. the Second European Conference on Multimedia Applications, Services and Techniques - ECMAST'97*, pp. 523-536, Milan, Italy, May 1997.
  - [9] J. Bolot, T. Turetti, and I. Wakeman, "Scalable Feedback Control for Multicast Video Distribution in the Internet," *ACM SIGCOMM*, vol. 24, no. 4, pp. 58-67, October 1994.
  - [10] J. Bolot and T. Turetti, "A Rate Control Mechanism for Packet Video in the Internet," *Proc. IEEE Infocom '94*, pp. 1216-1223, Toronto, Canada, June 1994.
  - [11] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet," RFC 2309, April 1998.
  - [12] I. Busse, B. Deffner, and H. Schulzrinne, "Dynamic QoS Control of Multimedia Applications based on RTP," *Second Workshop on Protocols for Multimedia Systems*, Salzburg, Austria, October 1995.
  - [13] A. Campbell, D. Hutchison, and C. Aurrecoechea, "Dynamic QoS Management for Scalable Video Flows," *Proc. Fifth International Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 107-118, Durham, New Hampshire, April 1995.
  - [14] A. Campbell, G. Coulson, and D. Hutchinson, "A Quality of Service Architecture," Technical Report MPG-94-08, Department of Computer Science, Lancaster University, 1994.
  - [15] CCITT, "Recommendation H.261: Video Codec for audiovisual services at p\*64 kbps," 1990.
  - [16] S. Cheung, M. Ammar, and X. Li, "On the Use of Destination Set Grouping to Improve Fairness in Multicast Video Distribution," *Proc. IEEE Infocom '96*, pp. 553-560, San Francisco, CA, March 1996.

- [17] D. Clark, S. Shenker, and L. Zhang, "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism," *Proc. ACM SIGCOMM'92*, pp. 14-26, September 1992.
- [18] D. Clark and D. Tennenhouse, "Architectural considerations for a new generation of protocols," *ACM SIGCOMM*, pp. 200-208, Philadelphia, September 1990.
- [19] T. Coreman, C. Leiserson, and R. Rivest, *Introduction to Algorithms*. McGraw-Hill and MIT Press, 1990.
- [20] G. Coulson, A. Campbell, P. Robin, G. Blair, M. Papathomas, and D. Hutchinson, "The Design of a QoS Controlled ATM Based Communication System in Chorus," Technical Report MPG-94-05, Department of Computer Science, Lancaster University, 1994.
- [21] G. Coulson, A. Campbell, P. Robin, and D. Shephard, "Supporting Continuous Media Applications in a Microkernel Environment," Technical Report MPG-94-16, Department of Computer Science, Lancaster University, 1994.
- [22] R. Cruz, "A Calculus for Network Delay, Part I: Network Elements in Isolation," *IEEE Trans. Information Theory*, vol. 37, no. 1, pp. 114-121, 1991.
- [23] S. Deering, "Multicast Routing in a Datagram Internetwork," PhD dissertation, Dept. of Computer Science, Stanford University, December 1991.
- [24] S. Deering and D. Cheriton, "Multicast Routing in Internetworks and Extended LANs," *ACM Trans. Computer Systems*, vol. 8, no. 2, pp. 85-110, May 1990.
- [25] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei, "The PIM Architecture for Wide-Area Multicast Routing," *ACM Trans. Networks*, April 1996.
- [26] A. DeSimone, R. Nagarajan, and Y.T. Wang, "Desktop and Network Performance Issues in Multimedia Conferencing and Collaboration," *Proc. PROMS'95*, April 1995.
- [27] S. Dharanikota, K. Maly, B. Kvande, C.M. Overstreet, and R. Mukkamala, "Providing predictable performance to a network-based application over ATM,"

- HPCS-95*, Mystic, August 1995.
- [28] Differential Services archives, <http://www-nrg.ee.lbl.gov/diff-serv-arch>,  
<http://diffserv.lcs.mit.edu/IETF40/index.html>.
  - [29] H. Eriksson, "Mbone: The Multicast Backbone," *Communications of the ACM*, vol. 37, no. 8, pp. 54-60, August 1994.
  - [30] D. Ferrari, A. Banerjee, and H. Zhang, "Network support for multimedia: a discussion of the Tenet approach," Technical Report TR-92-072, International Computer Science Institute, Berkeley, California, October 1992.
  - [31] D. Ferrari and D. Verma, "A Scheme for Real-Time Channel Establishment in Wide-Area Networks," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 3, pp. 368-379, April 1990.
  - [32] S. Floyd and V. Jacobson, "Random Early Detection gateways for Congestion Avoidance," *IEEE/ACM Trans. Networking*, vol. 1, no. 4, pp. 397-413, August 1993.
  - [33] S. Floyd and V. Jacobson, "Link-sharing and Resource Management Models for Packet Networks," *IEEE/ACM Trans. Networking*, vol. 3, no. 4, pp. 365-386, August 1995.
  - [34] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing," *ACM SIGCOMM*, pp. 342-356, August 1995.
  - [35] A. Gupta, W. Howe, M. Moran, and Q. Nguyen, "Resource Sharing for Multi-Party Real-Time Communication," *Proc. IEEE Infocom'95*, 1995.
  - [36] B. Haskell, A. Puri, and A. Netravali, *Digital Video: An Introduction to MPEG-2*. Chapman & Hall, 1997.
  - [37] F. Herrmann, F. Armand, M. Rozier, M. Gien, V. Abrossimov, I. Boule, M. Guillemont, P. Leonard, S. Langlois, and W. Neuhauser, "CHORUS, A New Technology for Building a Unix System," *Proc. EUUG Autumn Conference*, pp. 1-18, October 1988.
  - [38] C. Huitema, *IPv6 The New Internet Protocol*. Prentice Hall PTR, 1996.

- [39] D. Hutchinson, G. Coulson, and G. S. Blair "Quality of Service Management in Distributed Systems," Technical Report MPG-94-02, Department of Computer Science, Lancaster University, 1994.
- [40] V. Jacobson, "Congestion Avoidance and Control," *ACM Computer Communication Review*, vol. 18, no. 4, pp. 314-329, August 1988.
- [41] K. Jeffay, D.L. Stone, and F.D. Smith, "Transport and Display Mechanisms Across Packet-Switched Networks," *Computer Networks and ISDN Systems*, vol. 26, no. 10, pp. 1281-1304, July 1994.
- [42] X. Li and M. Ammar, "Bandwidth Control for Replicated-Stream Multicast Video Distribution," *Proc. IEEE HPDC-5*, Syracuse, NY, August 1996.
- [43] X. Li, S. Paul, P. Pancha, and M. Ammar, "Layered Video Multicast with Retransmission (LVMR): Evaluation of Error Recovery Schemes," *Proc. NOSS-DAV'97*, St. Louis, Missouri, May 1997.
- [44] X. Li, S. Paul, and M. Ammar, "Layered Video Multicast with Retransmissions (LVMR): Evaluation of Hierarchical Rate Control," *Proc. Infocom'98*, March 1998, San Francisco, CA.
- [45] K. Maly, H. Abdel-Wahab, C.M. Overstreet, C. Wild, A. Gupta, A. Youssef, E. Stoica, and E. Al-Shaer, "Interactive Distance Learning over Intranets," *IEEE Internet Computing*, vol. 1, no. 1, pp. 60-71, January 1997.
- [46] S. McCane, V. Jacobson, and M. Vetterli, "Receiver-driven Layered Multicast," *ACM SIGCOMM*, pp. 117-130, Stanford, CA, August 1996.
- [47] S. McCanne, M. Vetterli, and V. Jacobson, "Low-complexity Video Coding for Receiver-driven Layered Multicast," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 6, pp. 983-1001, August 1997.
- [48] D. Mitzel, D. Estrin, S. Shenker, and L. Zhang, "An Architectural Comparison of ST-II and RSVP," *Proc. IEEE Infocom'94*, Toronto, Canada, June 1994.
- [49] S. Pejhan, A. Eleftheriadis, and D. Anastassiou, "Distributed Multicast Address Management in the Global Internet," *IEEE Journal on Selected Areas in Communications*, pp. 1445-1454, October 1995.

- [50] H. Schulzrinne, "RTP: The real-time transport protocol," *In MCNC 2nd Packet Video Workshop*, vol. 2, Research Triangle Park, NC, December 1992.
- [51] H. Schulzrinne, S. Casner, R. Fredrick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC 1889, January 1996.
- [52] S. Senbel and H. Abdel-Wahab, "A Quadtree-based Image Encoding Scheme for Real-Time Communication," *Proc. IEEE International Conference on Multimedia Computing and Systems*, pp. 143-150, Ottawa, Canada, June 1997.
- [53] R. Steinmetz and K. Nahrstedt, *Multimedia: Computing, Communications and Applications*. Prentice Hall PTR, 1995.
- [54] W. Richard Stevens, *TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP and the UNIX Domain Protocols*. Addison-Wesley, 1996.
- [55] T. Talley and K. Jeffay, "Two-Dimensional Scaling techniques for Adaptive, Rate-Based Transmission Control of Live Audio and Video Streams," *Proc. ACM Multimedia'94*, pp. 247-254, San Francisco, CA, October 1994.
- [56] L. Vicisano and J. Crowcroft, "One-to-Many Reliable Bulk-Data Transfer in the MBone," *Proc. Third International Workshop on High Performance Protocol Architectures (HIPARCH'97)*, June 1997.
- [57] L. Vicisano, L. Rizzo, and J. Crowcroft, "TCP-Like Congestion Control for Layered Multicast Data Transfer," *Proc. IEEE Infocom'98*, pp. 996-1003, San Francisco, CA, March 1998.
- [58] W3C proposed recommendation, "Synchronized Multimedia Integration Language (SMIL) 1.0 Specification," <http://www.w3.org/TR/PR-smil>, April 1998.
- [59] M. Willebeek-LeMair and Z. Shae, "Videoconferencing over Packet-Based Networks," *IEEE Journal on Selected Areas in Communication*, vol. 15, no. 6, pp. 1101-1114, August 1997.
- [60] G. Wright and W. Richard Stevens, *TCP/IP Illustrated, Volume 2: The Implementation*. Addison-Wesley, 1995.
- [61] A. Youssef, H. Abdel-Wahab, and K. Maly, "Performance Evaluation of an Inter-Stream Adaptation Algorithm for Multimedia Communications," *Proc.*



*IFIP Conference on Performance of Information and Communication Systems (PICS'98)*, pp. 333-344, Lund, Sweden, May 1998.

- [62] A. Youssef, H. Abdel-Wahab, and K. Maly, "The Software Architecture of a Distributed Quality of Session Control Layer," *Proc. Seventh IEEE Symposium on High Performance Distributed Computing (HPDC-7)*, pp. 21-28, Chicago, IL, July 1998.
- [63] A. Youssef, H. Abdel-Wahab, and K. Maly, "A Scalable and Robust Feedback Mechanism for Adaptive Multimedia Multicast Systems," *Proc. IFIP Conference on High Performance Networking (HPN'98)*, Vienna, Austria, September 1998.
- [64] A. Youssef, H. Abdel-Wahab, and K. Maly, "Controlling Quality of Session in Adaptive Multimedia Multicast Systems," *Proc. Sixth IEEE International Conference on Network Protocols (ICNP'98)*, Austin, Texas, October 1998.
- [65] A. Youssef, H. Abdel-Wahab, K. Maly, and M. Gouda, "Inter-Stream Adaptation for Collaborative Multimedia Applications," *Proc. Second IEEE Symposium on Computers and Communications (ISCC'97)*, pp. 116-120, Alexandria, Egypt, July 1997.
- [66] A. Youssef, H. Abdel-Wahab, K. Maly, and C. Wild, "A Comparative Study of Two Generic Resource Allocation Models," *Proc. Third International Conference on Computer Science & Informatics (CS&I '97)*, pp. 47-50, Durham, NC, March 1997.
- [67] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A New Resource ReSerVation Protocol," *IEEE Network Magazine*, September 1993.
- [68] H. Zhang, "Service Disciplines for Packet-Switching Integrated-Services Networks," PhD dissertation, Dept. of Computer Science, University of California, Berkeley, 1993.
- [69] H. Zhang and D. Ferrari, "Improving Utilization for Deterministic Service in Multimedia Communication," *IEEE International Conference on Multimedia Computing and Systems*, 1994.

## APPENDIX A

### ANALYSIS OF THE A-IWFS ALGORITHM

In what follows, we compute the complexity and prove the correctness of the A-IWFS algorithm, which was presented in Section 3.6.

#### A.1 Complexity

The first phase of the algorithm includes a sort operation which is of  $O(N \log_2 N)$ . This phase is completed after the first  $N$  iterations of the **while** loop.

The second phase is composed of  $Ltot - N$  iterations of the **while** loop. In each iteration the minimum  $Sched_k \forall k$  is sought. The last iteration of these  $Ltot - N$  iterations requires 0 comparisons in the evaluation of the **Min** function. The iteration before last requires 1 comparison only, and so forth, whereas each of the first  $Ltot - 2N$  iterations requires  $N - 1$  comparisons. Thus, the total complexity for executing the **Min** function in the  $Ltot - N$  iterations is equal to  $0 + 1 + 2 + \dots + (N - 1) + (Ltot - 2N)(N - 1) = (N^2 - N)(a - 1.5)$ , where  $a = \frac{Ltot}{N}$  is the average number of layers per stream. Therefore, the complexity of this phase is  $O(N^2)$ .

The function **Recompute\_All\_Schedules** is called once for each stream, when the last layer of that stream is assigned an order. Thus, it is called  $N$  times, and the cost of its execution each time is  $O(N)$ .

From the above, we conclude that the total time complexity for executing the A-IWFS algorithm is  $O(N^2)$ .

## A.2 Correctness

We prove here that the algorithm meets the following design constraints:

1.  $i < j \Rightarrow Order_{k,i} < Order_{k,j} \forall k$ .
2. Share the available bandwidth fairly according to the weighted fair share policy.
3. Utilize all available bandwidth.
4. Maximize the number of admitted streams without violating the priority order.

### Proof

1.  $Lnext_k$  holds the next layer to process for stream  $k$ , and is only incremented by one each time a layer belonging to  $k$  is assigned its order. Thus for any two layers  $i$  and  $j$  belonging to the same stream  $k$ , if  $j > i$  then  $j$  cannot be assigned its order unless after  $i$  has been already assigned its order. Also,  $ord$  keeps track globally of the current position in the linear order, and is incremented by one immediately after assigning an order to one layer. Thus,  $j$  cannot be assigned an order which is smaller or equal to the order of  $i$ .
2. The algorithm assigns for stream  $k$  a weight  $w_k = \frac{p_k}{\sum_{i=1}^N p_i}$ , which is equivalent to its share of the bandwidth. As soon as a layer  $l$  for stream  $k$  is assigned the current position in the linear order, the next layer of  $k$  is scheduled to enter the linear order after  $Sched_k = \frac{1}{w_k} R_{k,l}$ . Thus  $Sched_k$  is set such that a fair share is given to every other stream before selecting the next layer from  $k$ . Since the layers are selected in order of minimum  $Sched_k$ , each stream is asymptotically assigned an amount of bandwidth equivalent to its weight.
3. The termination condition for the algorithm is when  $ord$  exceeds  $Ltot$ . Since  $ord$  is incremented by one only each time a layer is assigned its order, therefore the algorithm does not terminate unless after every layer is assigned an order. Also, whenever all layers from a source  $k$  are assigned orders, the value of  $tot.p$  is updated, by subtracting  $p_k$ , which in turn updates all the weights of the still non-fully

assigned sources. In addition, the function *Recompute\_AllSchedules* is invoked in order to immediately reflect the change in weights on the next order assignment. Thus, utilizing all the available bandwidth while the weighted fair share policy is in force among the non-fully allocated streams.

4. The first  $N$  layer assignments are done one per stream, in strict priority order, regardless of the values of  $Sched_i \forall i$ . This ensures maximization of the number of active streams without priority inversion, i.e., without turning off a higher priority stream to activate a bigger number of lower priority ones.

## APPENDIX B

### MONITORING AGENT CLIENT INTERFACE

The following is a listing of the API calls available to the monitoring agent client. The API is provided in C language.

#### B.1 Client Control Interface

**Monitor \*Q\_FlowSpec(Stream \*str)**

- **Description:** This call leads to the initialization of a Monitor structure, and establishment of communication between the monitoring agent and the ISA agent, if it does not already exist.
- **Arguments:** `str` is an already initialized Stream structure, possibly through `Q_UseFlowProfile()`.
- **Return value:** On success returns a pointer to the initialized Monitor structure. Otherwise, returns `NULL`.

**int Q\_UseFlowProfile(Stream \*str, int profileId)**

- **Description:** This function eases the process of initializing the Stream structure through the use of existing flow specification profiles.
- **Arguments:** `str` is a non initialized Stream structure, and `profileId` is the ID of an existing profile.

- **Return value:** On success returns 1. Otherwise returns -1.

**int Q\_InstallFlowProfile( (void \*) (ProfileFunc)(Stream \*str) )**

- **Description:** Installs a new flow specification profile.
- **Arguments:** ProfileFunc is a function that initializes str according to the new flow specification.
- **Return value:** On success returns a unique ID for the new profile. Otherwise returns -1.

**int \*Q\_InitMcastSend(Monitor \*mon, char \*addr, int ctlPort, int ttl, int loop)**

- **Description:** Initializes one or more multicast sockets to be used for sending data. In addition a control socket for sending/receiving state feedback protocol messages is initialized.
- **Arguments:** mon is a pointer to this stream's Monitor. addr is the IP multicast address of the base layer. The same address is used for control messages with a different port number. Consecutive IP addresses are used for the enhancement layers. ctlPort is the port number for exchanging control information. This number is unique per stream within a session and is used as an identifier for the stream. The port numbers for the data sockets are ctlPort+1 to ctlPort+n, where n is the maximum number of layers comprising the stream. ttl is the time-to-live. loop if equal to one, loop-back of multicast messages to the local host is enabled.
- **Return value:** Returns a pointer to an array of sockets for data and control messages exchange.

**int \*Q\_InitMcastRecv(Monitor \*mon, char \*addr, int ctlPort, int ttl)**

- **Description:** Initializes one or more multicast sockets to be used for receiving data. In addition a control socket for sending/receiving state feedback protocol messages is initialized.
- **Arguments:** `mon` is a pointer to this stream's Monitor. `addr` is the IP multicast address of the base layer. The same address is used for control messages with a different port number. Consecutive IP addresses are used for the enhancement layers. `ctlPort` is the port number for exchanging control information. This number is unique per stream within a session and is used as an identifier for the stream. The port numbers for the data sockets are `ctlPort+1` to `ctlPort+n`, where `n` is the maximum number of layers comprising the stream.
- **Return value:** Returns a pointer to an array of sockets for data and control messages exchange.

`int Q_Select(...)`

- **Description:** A wrapper for the unix `select(...)` system call. This allows the agent to monitor its control sockets in a way invisible to the client.
- **Arguments:** same as `select()`.
- **Return value:** same as `select()`.

`int Q_SetTimer(long tm, void(*AlarmHandle)(void *arg), void *arg)`

- **Description:** A wrapper for the UNIX `ualarm(...)` system call. This allows the agent to setup timers necessary for implementing the state feedback protocol, and for identifying the beginning/end of monitoring intervals, in a way invisible to the client.
- **Arguments:** `tm` timeout period in milli-seconds. `AlarmHandle` is the handler called when the alarm expires. `arg` is a pointer to any data structure that is passed as argument to `AlarmHandle()`.

- **Return value:** Unique identifier for the alarm if successful. -1 otherwise.

**int Q\_RemoveTimer(timerId)**

- **Description:** Stop a timer, and remove its entry from the timers queue.
- **Arguments:** timerId is a unique identifier of the timer that was obtained when Q\_SetTimer() was invoked.
- **Return value:** -1 on failure.

## B.2 Client Data Interface

**char \*Q\_GetBuffer(int size)**

- **Description:** This function allocates a message buffer space including the QoSess layer header space, and returns a pointer to the data segment of the message to the client. Clients should use only these buffers in sending/receiving data.
- **Arguments:** size is the data size requested.
- **Return value:** On success returns a pointer to the data segment of an allocated buffer. Otherwise returns NULL.

**void Q\_FreeBuffer(char \*buf)**

- **Description:** This function deallocates a message memory space including the QoSess layer header space.
- **Arguments:** buf is a pointer to the data area of the message requested to be released.

**int Q\_Send(Monitor \*mon,...)**



- **Description:** A wrapper for the UNIX `send()` system call. This allows the agent to update its status and to add monitoring information to the QoSess header of the sent message.
- **Arguments:** `mon` is a pointer to the stream's Monitor. The rest of the arguments are the same as `send()` original arguments.
- **Return value:** same as `send()`.

`int Q_Recv(Monitor *mon,...)`

- **Description:** A wrapper for the UNIX `receive()` system call. This allows the agent to update its status and to extract monitoring information from the QoSess header of the received message.
- **Arguments:** `mon` is a pointer to the stream's Monitor. The rest of the arguments are the same as `receive()` original arguments.
- **Return value:** same as `receive()`.

## APPENDIX C

### ACRONYMS

ATM	Asynchronous Transfer Mode
A-IWFS	Approximated - Iterative Weighted Fair Share
IETF	Internet Engineering Task Force
IMC	Interactive Multimedia Collaborative Application
IP	Internet Protocol
ISA	Inter-Stream Adaptation
I-WFS	Iterative - Weighted Fair Share
M-LBAP	Modified - Linear Bounded Arrival Processes
QoS	Quality of Service
QoSess	Quality of Session
RFC	Request For Comments
RISA	Rate-based Inter-Stream Adaptation
RSVP	Resource Reservation Protocol
RTP	Real-time Transport Protocol
RTT	Round-Trip Time
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

## VITA

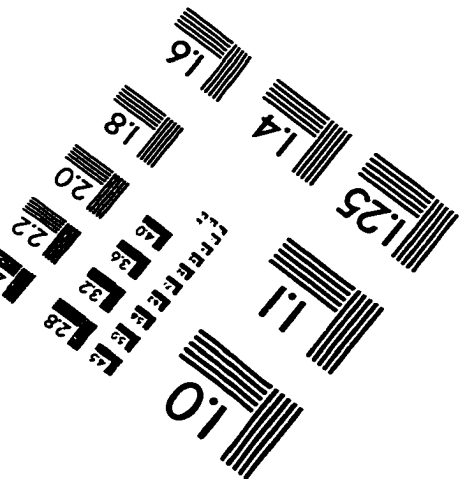
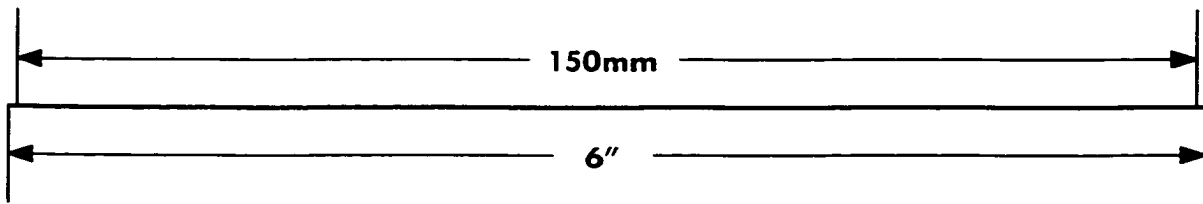
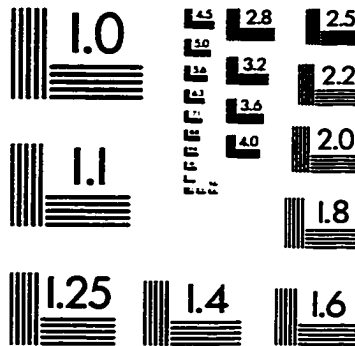
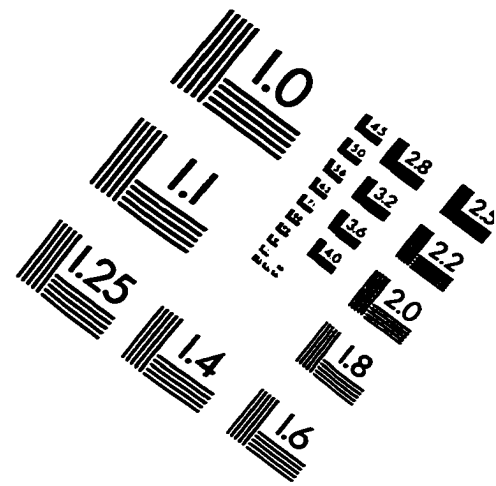
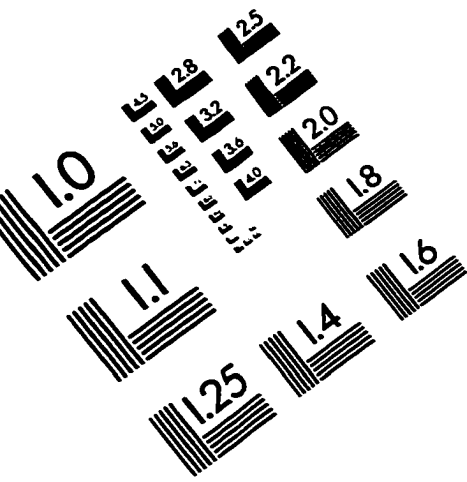
Alaa S. Youssef was born in Alexandria, Egypt, on November 23, 1968. He received his Bachelor of Science in Computer Science and Automatic Control from The Faculty of Engineering, Alexandria University, Egypt, in June 1991. He worked as a Teaching Assistant for the Department of Computer Science at Alexandria University from June 1991 to July 1994. In April 1994, he received his Master of Science degree from the same department. He started working on his Ph.D. Degree in Computer Science at Old Dominion University, Virginia, in August 1994. During the course of his Ph.D. work, he co-authored over twenty scientific papers and technical reports, and filed a patent for work done at IBM Thomas J. Watson Research Center during the summer of 1997.

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

This dissertation was typeset using  $\text{\LaTeX}^*$  by the author.

\* $\text{\LaTeX}$  is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's  $\text{\TeX}$  Program.

# IMAGE EVALUATION TEST TARGET (QA-3)



APPLIED IMAGE, Inc  
1653 East Main Street  
Rochester, NY 14609 USA  
Phone: 716/482-0300  
Fax: 716/288-5989

© 1993, Applied Image, Inc.. All Rights Reserved

