# Multi-party Real-time Communication in Computer Networks

by

Amit Gupta

Ph.D. Dissertation

Report No. UCB/CSD-96-896

Computer Science Division

University of California

Berkeley CA 94720

February 1996

amit@cs.berkeley.edu

# Multi-party Real-time Communication in Computer Networks

## Abstract

Multi-party Real-time Communication in Computer Networks
Amit Gupta
Doctor of Philosophy in Computer Science
University of California at Berkeley

The Internet has traditionally concentrated on availability: maintaining end-to-end connectivity in the face of unreliable systems and network congestion. Many emerging applications, however, require predictable performance and support for multi-party communication from the network service; at the same time, advances in networking technology have led to development and deployment of high-speed networks. The combined consideration of the stringency of the requirements of real-time communication and the high bandwidth provided by the new network technologies raise an interesting set of scaling and efficiency problems. This dissertation investigates mechanisms for supporting multi-party real-time communication in packet-switching networks.

We first present the work done by the Tenet Group at Berkeley in designing and building network protocols for supporting unicast real-time communication. These protocols serve as the framework for our research; given the framework, we describe a few salient issues that arise in multi-party communication: managing multicast group membership, supporting dynamic changes in these groups, and supporting heterogeneity in receivers. We then introduce the ideas that form the basis of the research effort: exploiting the characteristics of multi-party communication to improve the efficiency in using network resources; providing the network managers with effective ability to control the network resources; and providing a more usable service to the network users.

The traditional approach to supporting real-time communication allocates network resources to individual connections; this approach provides well-defined performance guarantees that are independent of other network traffic. To improve network resource utilization without sacrificing well-defined guarantees, we present *resource sharing*, which exploits relationships among connections to share resource allocations among them; the applications maintain complete control over the sharing as they explicitly specify these relationships. With resource sharing, for large conferences with a bounded number of concurrent speakers, resource requirements do not increase with the number of potential speakers. Therefore, resource sharing is an important tool for economically providing real-time performance guarantees for large conferences.

For real-time communication services to achieve widespread usage, it is important that the protocols and schemes provide good capability for the network's management to control the allocation of resources. For this capability, we present *resource partitioning*, i.e., distributing the different resources available at any system among a number of partitions. Resource partitioning can then be used to form virtual private sub-networks. These sub-networks have many applications: the network management can keep a small fraction of resources for management and fault-handling traffic, or for non-real-time traffic; and better support for mobile computing and for advance reservation of real-time connections.

Conferencing and other important distributed multi-party multimedia applications would benefit from a network service that provides support for advance reservations. The network service clients who wish to set up multimedia multi-party meetings need to schedule those meetings in advance to make sure that the participants will be able to attend, and would like to obtain assurances that the network resources will be available for the entire duration of the meeting. We have devised mechanisms for reserving resources for real-time connections in advance.

We have devised mechanisms for resource sharing, resource partitioning, and advance reservations; it is critical that these mechanisms work well together, and with other components of our real-time communication system (e.g., routing). We have designed and implemented the Tenet Protocol Suite 2, which incorporates these mechanisms to provide network support for multi-party real-time communication. Simulation results show that these mechanisms interact well with one another; preliminary results from a measurement study show that the protocols are effective in supporting guaranteed performance multi-party applications in an internetworking environment.

*To my parents, Shakuntala and Rameshwar*

# Contents

# List of Figures

## Acknowledgments

I would like to express my earnest gratitude to my advisor, Professor Domenico Ferrari, for his guidance and support throughout my research. he provided me a lot of freedom to explore and investigate the research topics and ideas that I found interested, and yet he was always available for providing useful direction and advice when I needed; I am very impressed with the mastery with which he maintained the delicate balance between these two.

I thank the members of my thesis committee, Professor Randy Katz and Professor Charles Stone for reading my thesis and making suggestions for improvement. I am also thankful to Professor Lawrence Rowe for chairing and to Professor Jean Walrand and Professor Charles Stone for serving on my quals committee.

It has been a great experience for me to work in the Tenet group; I want to thank Anindo Banerjea for advising me to join the Tenet group. Collaborations with other group members have been a vital component in my research throughout my Ph.D years. My first project in the group was with Anindo Banerjea on bandwidth-delay bound trade-offs in round-robin schedulers; I then worked with Ramon Cecares on congestion control in high-speed ATM networks, and with Mark Moran and Professor Bernd Wolfinger on Continuous Media Transport Protocol (CMTP). I would like to thank them for the invaluable experiences and knowledge that they imparted to me. I also benefited from the previous research by the Tenet Suite 1 designers and implementors; in particular, Hui Zhang and Dinesh Verma designed RTIP, while Anindo Banerjea and Bruce Mah designed RCAP for Tenet Suite 1; I want to thank them for sharing their expertise with me.

These projects introduced me to the networking research, and finally led me to the current thesis topic. In incorporating these ideas in the Tenet protocols (Suite 2), I benefited from discussions with many Tenet members and visitors; these researchers made several significant contributions to the Tenet Suite 2 project under Professor Ferrari's guidance and leadership. Key contributors to Suite 2 design and implementation include Riccardo Bettati, Wolfgang Effelsburg, Wendy Heffner, Winnie Howe, Rebbie Moon, Mark Moran, Eberhard Mueller-Menrad, Quyen Nguyen, Jean Ramaekers, Paola Rossaro, Clemens Szyperski, Giorgio Ventre, Ron Widyono, Raj Yavatkar, and Makiko Yoshida.

I also enjoyed working with Professor Kurt Rothermel on designing fault detection and recovery mechanisms for real-time communication, with Rolf Oppliger in designing security schemes for Tenet Scheme 2, with Margaret Tran on handoff rerouting algorithms for wireless communication. I also thank Winnie Howe and Quyen Nguyen for helping me with resource sharing simulations; it gave me great joy to work with such talented and competent researchers. I also gained a lot from my discussion with many other Tenet members and ICSI researchers; I would like to especially thank Andres Albanese, Luca Delgrossi, Rahul Garg, Riccardo Gusella, Steve McCanne, Vern Paxson, Fred Templin, and Dinesh Verma for many interesting, useful, and intellectually stimulating discussions.

I also want to thank Chuck Kalmanek and Srinivas Keshav of AT&T Bell Laboratories, Lixia Zhang and Scott Shenker of XEROX Palo Alto Research Center, Sally Floyd of Lawrence Berkeley Laboratories, Andrew Campbell of Columbia University, Professor

Doug Shephard of Lancaster University, and Professor Lawrence Landweber of University of Wisconsin for many useful discussions, and for being genuinely interested in my research. Special thanks go to Roya Ulrich for reading my thesis and providing many useful suggestions.

Finally, I want to thank my family; even though they have been on the other end of the world all this time, they have provided me with extremely valuable emotional support and guidance; I could not have finished this thesis without their support.

# Chapter 1

# Introduction

The increasing speed and connectivity of computer networks and the improvement of workstation capabilities are enabling a new class of distributed applications. The current trend is to use the new high-speed networks for the services that had previously required special, dedicated networks, while continuing to provide also the more traditional data communication services. Of special interest are the services for supporting distributed multi-party real-time communication applications[1] such as distributed audio and video conferencing, distributed classrooms, virtual meetings and electronic town halls. Some less common, but nevertheless important, applications include real-time monitoring and control, scientific visualization and medical imaging and collaboration applications [110, 13, 71, 84, 109, 48]. It is widely believed that these applications should be supported in the general framework of real-time communication[18, 40, 49, 84, 74]. Real-time communication requires predictable performance (e.g., the end-to-end data delivery delay be bounded); typically, the network service clients negotiate with the network service provider to obtain a desired Quality of Service (QoS), which the network service provider guarantees[4, 114, 106, 97, 31, 43].

The combined consideration of real-time and multi-party communication opens an interesting area of research that is highly relevant for emerging multimedia conferencing applications. The task of designing efficient and scalable schemes for providing the network support for these applications is a challenging one. The stringency of the requirements of real-time communication and the high bandwidth provided by these new networks raise an interesting set of scaling and efficiency problems.

This dissertation is about mechanisms for supporting multi-party real-time communication in these high-speed, integrated-services networks. In this introductory chapter, we will first provide the background information: real-time communication, Integrated Services Packet Switching networks, and the first set of Tenet algorithms and mechanisms (Tenet Scheme 1) and the protocols that embody these techniques (Tenet Protocol Suite 1). We will then describe the key issues in multi-party communication along with the key network service management issues that arise in multi-party environments and conclude this chapter with a brief overview of this dissertation.

[1]The term multi-party refers to applications with more than two participants, and the term real-time implies that the network clients require performance guarantees from the network service provider.

## 1.1 Background

In this section, we provide the background of our investigation by first discussing two of the keywords in the title of this dissertation: Integrated services packet switching networks, and real-time communication. We also briefly describe the first Tenet protocols, which supported *unicast* real-time communication; our work, together with the work of others, extends these protocols to multi-party environments.

### 1.1.1 Integrated services packet switching networks

Traditionally, two different types of networks have supported the communication needs of the users: the telecom networks (including the telephone, cable, and satellite networks), and the data networks (e.g., the Internet). The telephone networks were designed (and used) primarily to support high quality audio communication between telephones; thus, these networks were carefully engineered to provide low delay and fixed-bandwidth service. On the other hand, the data networks (including the Internet) were designed primarily to support data exchange between computers. These networks typically support a spectrum of networking protocols[2]; these protocols offer services to facilitate easy exchange of information over a variety of networks and networking technologies. These protocols have traditionally provided a "best-effort" service; they do not make any performance guarantees (unlike the fixed-bandwidth service offered by the telecom networks) except that they (and the networks) will try their best to transmit the data through the network as long as there exist some viable path(s) between the data senders and receivers.

To support these disparate service models, these networks have traditionally relied on different underlying technologies. Traditionally, the telecom networks are based on "circuit-switching"; in circuit-switching, at the start of the conversation (*connection establishment time*), the network determines the connection route, as well as establishes an end-to-end physical "copper" path from the sender to the receiver. When a connection set-up signal goes through an intermediate node ("switch"), a physical connection is (conceptually) established between the input line and one (or more) of the output lines. On the other hand, the data networks are usually based on "packet-switching". Here, no physical "copper" path need be established between the sender(s) and the receiver(s). Instead, when the sender has a block of data to send, it sends it to a nearby switch, which receives the data block ("packet") in its entirety, and then forwards it to the next switch; the switch may also inspect the packet for possible errors.

The digitization of telephony and the increasing use of multi-media in distributed computing applications has led to the convergence o of computing and communications; this convergence has led to the emergence of a single, unified network, the Integrated Services Packet Network (ISN). Using a single unified telecom infrastructure offers many advantages; of critical importance are the vast economies of scale, and potential support for "integrated application" that can benefit from the synergy of this convergence.

[2]A protocol is a set of rules that dictate how computers communicate and exchange information over a network.

Many important technical (as well as non-technical) issues must be addressed before we can reap the benefits of this convergence. In particular, we need to support "telephony-like" applications (for example, distributed tele-conferences with multimedia, medical imaging and collaboration, and so on) in networks that must also support the traditional data applications equally well. To maintain the statistical multiplexing gains as in traditional data networks, the new *integrated services* networks should also use packet-switching[3]; on the other hand, the networks must provide better support for distributed multimedia and related applications ("real-time" applications). These considerations have led to some very interesting research in supporting real-time applications in Integrated Services Packet Networks; we discuss some of the issues in supporting real-time communication in the next sub-section.

### 1.1.2 Real-time communication

We now describe some of the principles that embody the Tenet view of the desirable characteristics of a solution to the problem of real-time communication [44]; in this view, continuous-media (digital audio, digital video, and so on) communication at the network and transport layers is considered to be a special case of real-time communication. The key property of real-time communication is that the clients require predictable performance, and are sensitive to loss of performance (for example, lost packets, variation in delays in transmitting data, and so on); to provide predictable performance, the network should offer *a priori* guarantees to the client and ensure that the guarantees cannot be violated. For providing these guarantees, the clients must specify their performance requirement to the network; in the current work, all the network-oriented requirements of continuous media are expressed in terms of bounds on the performance and reliability of the network (e.g., bounds on end-to-end delay, delay variation, and so on.; we refer to this set of bounds as Quality-of-Service (QoS) parameters).

During the design and implementation of the Tenet Protocol Scheme 1 for unicast (i.e., single sender, single destination) real-time communication, the Tenet Group proposed, and adopted, the contract model of client-service negotiations. In this model, at the connection set-up time, the client provides the network service provider with two sets of parameters: the traffic specification and the performance specification; the network service (after performing some internal, possibly distributed, computations) can *accept* or *refuse* the client's request. If the network accepts the contract, then it promises that the data transmission service will meet (or exceed) the client-specified performance bounds specified in the contract, *as long as* the client-generated traffic remains within the traffic bounds (as specified in the contract). If the client traffic violates the traffic bounds, then all bets are off, and the network is free to violate the performance bounds, dropping packets if it so pleases.

Now, these contract-specified QoS guarantees cannot be provided if the network does not check for saturation before accepting new connections. As the load on the network rises, a point is reached where the admission of a new connection would overload the

[3]For the purpose of this dissertation, we shall use the term packet-switching to also include cell-switching; in cell-switching, all packets (or cells) have the same size.

system and prevent the network from meeting the performance requirements of this new connection, or of one of the already established connections, from being satisfied. Thus, the network must perform *admission control* to ensure that guarantees are met. Another way of viewing admission control is that the network is allocating ("reserving") resources for the guaranteed-performance connections. Note, however, that reservation does not imply that the resources remain unused if the reserving client can not use them. In keeping with the principles of packet-switching and statistical multiplexing, the reserving client only has the first priority in using these resources; if the reserving client does not have data to send, these resources can (and should) be used by other clients.

The resources (e.g., link bandwidth, node buffers etc.) are reserved in the network on all the nodes that lie on the data transmission path ("route") of a real-time conversation. This reservation requires knowledge about the client QoS requirements and the connection's traffic characteristics. These characteristics are associated with the lifetime of the connection, and, consequently, so is the lifetime of the resource reservation requirement. Real-time guarantees can only be met for connections for which resources can be reserved.

Thus, this reservation implies a connection-oriented approach, in which the data transfer is preceded by a connection establishment phase. During this phase, resources are reserved on all nodes on the path of the connection. These resources are released at the end of the conversation, and the resource release process is called the connection tear-down process.

Another important component of our approach is the protection of real-time connections. Note that the contract specifies that the network service will meet the given performance bounds as long as the client traffic remains within the traffic specification, *irrespective of the behavior of other clients*. Now, this performance cannot be guaranteed if the connections are not protected from misbehaving (or malicious) real-time and non-realtime sources. To avoid violating the guarantees made to real-time connections, the network must either explicitly control the input rates on a per-connection basis, or adopt scheduling algorithms that will do so automatically in the nodes (e.g., Fair Queueing [38], Weighted Fair Queueing [89, 90, 88], RCSP [125]).

Based on these principles, the Tenet Group designed the Tenet Protocol Scheme 1 for unicast real-time communication; we describe selected salient features of the Tenet Scheme 1 (and its embodiment in the Tenet Protocol Suite 1) in the next subsection.

### 1.1.3 The previous Tenet Protocol Suite

Since 1987, the Tenet Group has worked in providing real-time communication in computer networks. This research led to the design and development of algorithms (Tenet Scheme 1) and their embodiment in the Tenet Protocol Suite 1. In this section, we will briefly describe the key components of the Tenet Scheme 1, which provides support for simplex, unicast (i.e., single sender, single destination, unidirectional data flow) real-time communication in packet-switching networks.

The key abstraction in the Tenet Scheme 1 is a *real-time channel*: an end-to-end simplex, unicast date connection characterized by traffic and performance bounds specified by the client. The traffic values provide an upper bound to the rate at which the sending client may put the data on the network; this traffic specification may be in terms of the Xmin-Xavg-I-Smax model [117]. The performance specification for a channel includes bounds on end-to-end delay (*delay bound*), the variation in the end-to-end delay (*jitter bound*), and the probability that end-to-end delay experienced by a packet will be within the delay bound. As per the contract model, if the network service *accepts* a client's request to establish a channel, it must provide performance at least as good as specified by these performance parameters, as long as the channel traffic remains within the specified traffic bounds, regardless of the behavior of other clients.

The Tenet approach is connection-oriented and reservation-based: before a real-time channel can be used by its requester, it must be established, (i.e., resources for the channel must be set aside along its route) so that the guarantees are supported. A key feature of the Tenet protocols is the separation of data delivery and control: RCAP provides signaling and RMTP/RTIP support data delivery for real-time communication[4, 3, 129, 126, 124, 46]. These protocols co-exist with the traditional Internet protocols (TCP,UDP,IP); indeed, over an internetwork, RCAP can use TCP/IP for reliable transmission of signaling messages. Figure 1.1 shows the Tenet protocol stack.

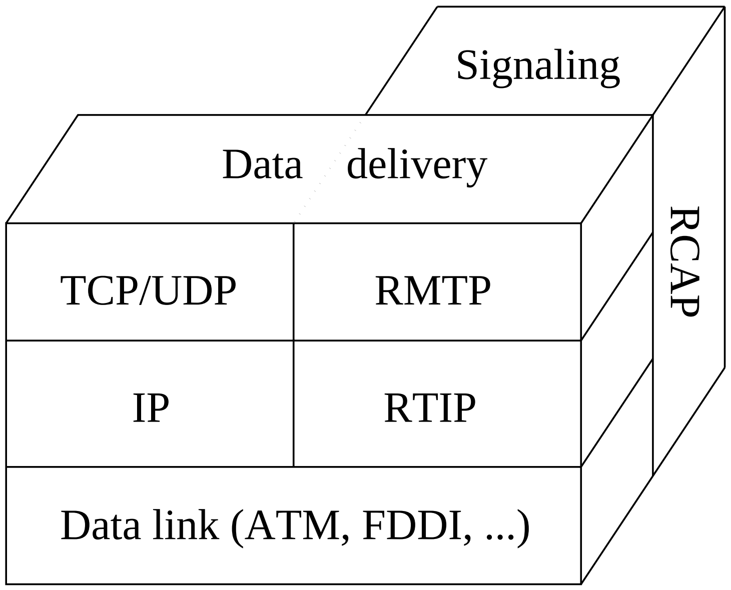

Figure 1.1: The Tenet protocol stack

The Real-time Channel Administration Protocol (RCAP) supports channel set-up, teardown, and other administrative tasks in response to requests from applications (and possibly to changes in the network state, e.g., node or link failure). For channel set-up, RCAP communicates with RTIP entities at each node along the channel's path.

The Real-Time Internet Protocol (RTIP) provides connection-oriented, performance guaranteed, unreliable, sequenced delivery of packets. Its services are used by the Real-time Message Transport Protocol (RMTP) which provides connection-oriented, performance guaranteed, unreliable, sequenced delivery of messages.

Channel establishment (with RCAP) is a distributed process; a message issued by the RCAP daemon at the source (*Establishment-message*) visits each node (switch, router, gateway) along the route of the channel. This message causes several admission tests and computations to be performed at the corresponding RCAP daemon at each node. If the new channel passes all the tests in a node, the message is forwarded, with some state information about the current node, to the RCAP daemon at the next node on the route. The final tests are performed by the destination; if they are successful, a channel-establishment message is sent by the destination RCAP daemon to the RCAP daemon at the source along the reverse route; when each node is revisited, the message corrects the tentative reservations made in that node by the forward message, and, on receipt of this message, the RCAP daemon informs the RTIP entity at that node about the performance bounds assigned to it. Some of these bounds will be used by RTIP during data transfer operations, others by RCAP daemon in tests for the admission of future channels, and some in both types of circumstances. If any of the tests in the nodes or in the destinations fails, the channel cannot be established, and a channel-reject message is immediately sent back to the source RCAP daemon. This messages removes in each node it visits the tentative reservations made for the new channel. This is shown in Figure 1.2.

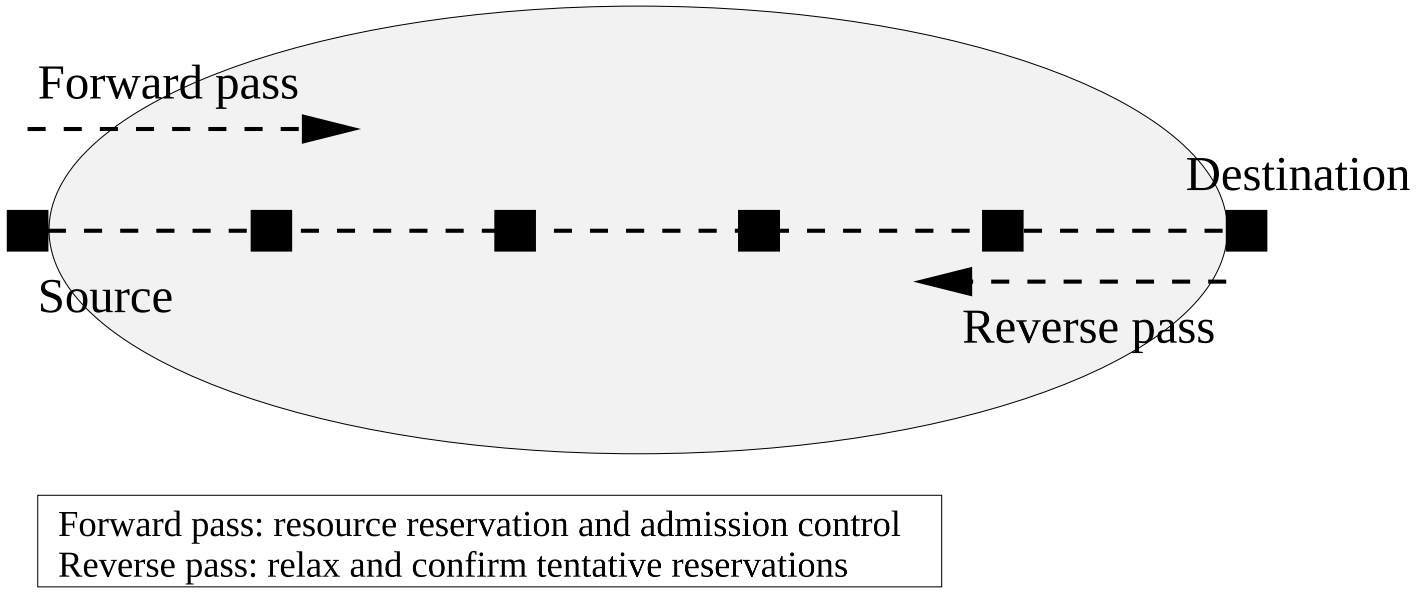
Forward pass
Destination
Source
Reverse pass
Forward pass: resource reservation and admission control
Reverse pass: relax and confirm tentative reservations

Figure 1.2: Two-pass channel establishment

A new channel's route may be computed by the source (or destination) if this host has the necessary topological information; knowledge by this host of the current real-time load (i.e., a measure of how much of each resource is currently earmarked for use by a real-time channel), and of such additional information as propagation delays and error rates of the links involved, is also quite useful to increase the probability that the chosen route will be able to support the new channel. An alternative is the construction of the route in a hop-by-hop fashion, with individual nodes usually not knowing much beyond the real-time load of their immediate neighbors.

To complete this general description of Scheme 1, we only have to mention the two special aspects of data transfers: scheduling and distributed rate-control. Most scheduling policies can be used for real-time communication, under fairly liberal conditions [42]. For this description, we can assume the Multi-class Earliest-Due-Date (EDD) discipline in its two versions: the so-called Delay-EDD (or D-EDD) [49, 127] and Jitter EDD (or J-EDD) [42, 127]. Rate control, either at the periphery of the network or in all of its nodes, is needed to protect well-behaving channels from the misbehavior of faulty or malicious sources.

## 1.2 Multi-party communication

In the previous section, we described the key issues and concerns in unicast real-time communication and how these were addressed in the design of the first generation of Tenet protocols. In this section, we first discuss some aspects of the service that the network must provide, including supporting heterogeneity and decoupling senders and receivers.

In Section 1.2.1, we will briefly describe some of the key ideas that led to the solutions proposed to address these concerns. Also, the network protocols must address some network management concerns, especially w.r.t resource allocation policies, fault-handling, and security; we will discuss these issues in Section 1.2.2. Throughout this discussion, we will use a distributed video-conference as the canonical example of multi-party real-time communication.

### 1.2.1 Key issues in multi-party communication

In this section, we discuss some of the issues in the design of network services for multi-party communication; these issues arise primarily due to the multi-party nature of the communication.

**Multicast groups:** Many multi-party applications involve a large number of recipients for each data stream; it is clear that multicasting can be used to reduce the traffic on the network nodes and links, thereby saving valuable network resources.

Now, a key component of the multi-party communication is the presence of multiple senders and receivers. A strawman multicast scheme would require that, at connection establishment time, the sources specify the list of receivers. It is unreasonable to require that in a large-scale distributed multimedia application (e.g., computer-based conferencing) the sender (or for that matter, any central application-based authority) know about all the receivers; it is equally unreasonable to require the receivers to know about all potential senders for that conference. It is important that the network service support this decoupling between the different participants; the network should provide the rendezvous among the participants interested in a common session.

The real-time nature of the conference also favors this separation of the senders and receivers. It is expected that receivers will be heterogeneous, i.e., that they will vary in their ability to handle the data, and the QoS requirements that they may have. It is generally unreasonable to expect the senders to specify these properties for all possible destinations of their data stream; this will also not scale well to very large conferences.

Also, multi-party conferences tend to be long-lived; the presence of multiple senders and receivers raises another issue: the membership in a multicast group may be dynamic, i.e., receivers may join to listen to (or leave) a session while it is in progress, and previously passive participants may become active, i.e., they may start sending data. It is important that the network service provide support for dynamic changes in group membership, without disrupting the "in-progress" conference.

For supporting the above-mentioned aspects of multi-party communication, the key abstraction is the *real-time multicast group*, for which we use the term "*Target set*". The Target set abstraction is the real-time analog of the IP Hostgroup abstraction, in that, while an IP Hostgroup has, as members, the destinations interested in listening to a common session, the Target set members are the interested destinations along with the requested bounds on end-to-end performance (e.g., end-to-end delay, jitter, i.e., variation in the delay, and so on). A channel logically transmits data from a particular sender to a Target set; this amounts to transmitting the data from that sender to all members of the Target set. Receivers can dynamically join and leave a Target set; when they join a Target set, they start getting data on all channels sending data to the Target set. In this manner, the Target sets support the decoupling between the senders and the receivers and also provide a rendezvous mechanism among them.

**Supporting heterogeneity**: Within one distributed video-conference, the different participants may differ significantly in many respects: the different senders may use different video/audio encoding schemes, the media streams may have very different data rates, the different receivers may vary in their ability to handle the data, e.g., due to different computing powers for processing the data streams and/or different capacities for displaying video data (for example, color vs. monochrome displays). Some participants may be much further away from the "group" as compared with other participants (for example, most participants may be based in California while a few participants may be in Japan); these differences would lead to different end-to-end delays to different receivers (from the same sender) as well as from different senders (to the same receiver). Different links in the network may differ substantially in latency and available bandwidth. We should not expect all receivers to request (or require) the same quality of service from the network; the network service should efficiently and effectively support large degrees of heterogeneity in the participants, in the data streams, as well as in the underlying networking technologies and link capacities.

For supporting heterogeneity in performance requirements (e.g., end-to-end delay bounds), the network service should permit different receivers (within the same multicast group) to independently specify their performance requirements. Also, for effectively supporting heterogeneity in link bandwidth in the network, as well as in the display equipment and other computing resources (for example, hardware support for video decoding), it is desirable that the senders use hierarchical (or layered) coding[113, 55, 107]. The senders can then send the data streams (corresponding to the different layers) on different multicast groups. For example, with two groups, the base layer can be sent to the base Target set, and the higher layer sent to the optional Target set. All receivers will first attempt to get the base layer (by joining the base Target set). If they want higher resolution, they can then attempt to join the optional Target set. In this way, receivers along the low-bandwidth paths will get the base layer, while the receivers along the high-bandwidth paths will get the full video stream [80, 115, 29].

**Resource sharing:** Traditional real-time network systems (e.g., [44]) treat traffic on different connections independently when determining their resource requirements; for multi-party real-time communication, this results in inefficient over-allocation of resources [62]. For example, consider an audio conference of one hundred persons. In a strawman proposal, the conference is set up by establishing one hundred multicast channels, one from each speaker (sender) to all listeners (destinations). It is reasonable to expect that only one person speak at any time. Along common sub-paths (for these hundred channels), it would be sufficient to reserve resources for two audio channels (to allow some over-speaking). Unfortunately, as per the traditional approach, if fifty of these channels overlap along some common sub-path, the network would reserve enough resources for fifty audio channels; this is clearly wasteful over-allocation. The resource allocation can be reduced (and the allocation efficiency increases) if the network clients can specify these *resource sharing* properties to the network, and if the network can use such information to reduce the resource allocation along common sub-paths.

We have devised the *channels groups* abstraction for the network clients to inform the service about such sharing (and other similar) relationships among different channels [63, 64]; the *resource sharing* channel groups allow the network clients to specify these resource sharing relationships to the network [62]. In the above example, the application would: (a) create a new channel group, and (b) ask the network to include the hundred audio channels in this channel group. The client would also inform the network that, at any server in the network, the aggregate resource allocation for all channels should not exceed two audio channels. During channel establishment for these channels, at any server, the admission test system can determine if it has already allocated resources for two audio channels, and, if so, accept this new channel without allocating any more resources. This mechanism is fully distributed; different servers make this decision independently. Chapter 2 provides an in-depth investigation of resource sharing.

**Advance reservations:** Conferencing and other important distributed multi-party multimedia applications would benefit from a network service that provides support for advance reservations. The network service clients who wish to set up multimedia multi-party meetings need to schedule those meetings in advance to make sure that the participants will be able to attend, and would like to obtain assurances that the network connections and the other required resources will be available for the entire duration of the meeting. In Chapter 4, we will describe the mechanisms that we designed for the Tenet Scheme 2 to provide its users with the ability to book network resources (far) in advance of their use[47]; this advance booking requires long-lived state in the network and it thus raises some interesting questions. How is this state stored? If a link goes down, should we also reroute the advance-reserved channels that are to traverse this link in the distant future? Do we need separate mechanisms for handling advance reserved channels, or can we effectively re-use mechanisms designed for non-advance channels? These issues are addressed in Chapter 4 as well as [64].

### 1.2.2 Network service management related issues

For real-time communication services to achieve widespread usage, it is important that network managers be allowed to control the services effectively, and that the network clients be able to usefully and effectively utilize the services offered. In this section, we will talk about three issues that impact the usefulness of the service provided: controlling resource apportionment, handling failures, and security.

**Resource partitioning**: An important capability concerns resource partitioning, i.e., distributing the different resources available at any given server (network node or link) among a number of partitions. For a given connection, the admission control and establishment computations are completely independent of the connections accepted outside the partition the connection belongs to. This independence amounts to splitting the server into a number of sub-servers, where the QoS guarantees can be made to the clients by only considering the connections belonging to that partition; yet, the promises are valid as long as the admission tests and rate control schemes for other partitions behave correctly, and these promises are independent of the individual per-connection establishment decisions and computations performed in other partitions.

This independence is very useful. The different sub-servers can be put together to form virtual private sub-networks. The network's management can keep a small fraction of resources for management and fault-handling traffic. Another possible application is that a fraction of the network resources be kept for non-real-time traffic. Other applications include fast establishment of real-time connections, support for mobile computing, and advance reservations of real-time connections.

In Chapter 3, we will describe the techniques that we have devised for resource partitioning; these techniques require changes only in admission control tests; the per-packet scheduling discipline remains unchanged. This technique works within the context of the Tenet protocol schemes: per-partition tests and computations have been shown to be derivable from those that apply to the entire network in each node; therefore, the subnetworks defined by partitions of each node's resources can be treated during channel establishment and teardown as though they were independent and isolated networks. The amount of a resource assigned to a given partition in a node may differ from the amounts assigned to the same partition in other nodes.

**Failure recovery**: The failure-handling sub-system is an integral component of any real-time communication service; indeed, for an operational network, it is critical that the network services behave gracefully when any component fails. While other researchers have previously considered failure-handling for non-real-time communication as well as for unicast real-time communication, these failure-recovery techniques must be reexamined in the light of the changes introduced by the new protocols and services for supporting multi-party real-time communication. In [64], we describe techniques and mechanisms for maintaining network services for multi-party real-time communication in the face of failures that may make parts of the network inaccessible. The key goal is that the protocols should provide high performance during normal operations (i.e., in the absence of failures), and the network performance should gracefully degrade in the face of network failures; e.g., in the presence of failures, the routes selected may not be optimal, connection set-up may take a little more time, or resource allocation may be less efficient. This is achieved by setting appropriate policies for storing state information in the network, as well as mechanisms for re-establishing connectivity for previously established connections and to set up new connections to existing conferences. [64] also describes a redundancy-based approach, using forward error correction (FEC), and dispersing the FEC'ed data among disjoint routes. With these mechanisms, we can make multi-party real-time communication protocols robust to single and/or multiple failures in the network, *without* diluting the strength of the performance guarantees offered, or sacrificing the system performance in the common case, i.e., when all components work correctly. These failure recovery mechanisms were designed to work with the multi-party real-time communication techniques proposed and described in this thesis.

**Security**: For an operational network, another key consideration is security; the security concerns encompass several related issues. First, the network must prevent unauthorized use of its resources (where such use may prevent legitimate users from utilizing the network services). Second, the network must ensure that malicious and/or mis-behaving clients not be able to disrupt the service of other, conformant clients. Third, the clients should be able to control data reception, in that unauthorized users should not be able to listen to other users' conversations. Also, the receivers should not receive data other than that sent by authorized senders. Also, such security mechanisms must scale well to large conferences, and it is desirable that they work even when parts of the network may not be available. Last, but perhaps most importantly, such security measures should not adversely impact the richness or the quality of service offered by the network, and the overhead (of security mechanisms and related computations) should not significantly impact the overall network performance.

In [85], we describe the security mechanisms that we designed for Tenet Scheme 2; these mechanisms were designed to work with the multi-party real-time communication techniques proposed and described in this thesis. These mechanisms are based on network-generated pseudo-random keys (public-key cryptography) associated with each object (Target Set, Channel, Group) in the network; these keys are provided to the object creator which can pass these "*capabilities*" around to other authorized users; these mechanisms also support operations on these keys, to enable the authorized users to pass *restricted* privileges to other users.

## 1.3 Thesis overview

In this research, we study the issues and tradeoffs that impact the design of network services to support multi-party real-time communication in integrated-services packet-switching networks. We adopt the following research methodology: (a) determine first the requirements for the network service, i.e., what are the services that the users will require or benefit from; (b) determine the key properties of multi-party communication; (c) design new mechanisms that exploit these properties of multi-party real-time applications to better support the needs of these users; (d) evaluate the proposed mechanisms via analysis and simulation; and finally, (e) implement these new technologies and evaluate their performance in real-time environments.

Chapter 2 presents resource sharing; resource sharing is based on the simple observation that in multi-party conferences, the participants usually co-operate. This co-operation can be exploited to reduce the network resource allocation for such multi-party applications; this increased resource allocation efficiency is critical for supporting large conferences. In the chapter, we first present a simple example that illustrates this co-operation; we then describe the three components of the resource sharing mechanisms: the client-service interface for the applications to explicitly specify potential resource sharing, the changes in the resource reservation and admission control system to support resource sharing (during the connection set-up phase), and the changes in the data delivery protocol to ensure that the applications' real-time guarantees are met even when other applications use resource sharing.

We evaluated the performance gains due to resource sharing by analysis as well as by simulation. The analysis provides useful lower bounds on performance gains in many different cases, including sparse networks as well as dense networks with bottlenecks and hot-spots. The analysis also shows that the routing protocols can significantly impact the resulting resource sharing gains; dynamic, load-balancing routing algorithms can reduce the resource sharing gains, while a "sharing-aware" routing system can significantly improve the resource sharing gains. The simulation results quantified the resource sharing gains and confirmed the analytical results: resource sharing is very useful in reducing network resource allocation.

Resource sharing is a critical and integral component of our multi-party real-time communication system; it is critical that resource sharing mechanisms work well with the other components of the full system; we conclude Chapter 2 by discussing these interactions.

Chapter 3 presents resource partitioning. For operational networks, it is important that the network service managers be able to control the distribution and apportionment of network resources among groups of users and/or classes of application; our resource partitioning mechanisms provide such capability to the network service providers. Our resource partitioning techniques work at the connection set-up and resource reservation stage; they do not affect the data delivery protocols at all. Since the resource reservation and admission test algorithms at a network server depend on the packet scheduling discipline followed there, the corresponding *partitioned* admission tests also depend on the packet scheduling discipline. In this chapter, we provide, with proofs, the admission tests with resource partitioning, for a spectrum of packet scheduling disciplines, including Earliest-Due-Date (EDD) [49], First-In-First-Out (FIFO) [125], Rate-Controlled-Static-Priority (RCSP) [125], and Weighted-Fair-Queueing (WFQ) [88].

We evaluated the performance of our resource partitioning mechanisms via simulation[4]; the simulations show that our techniques are useful and efficient, and the mechanisms work well. The *resource allocation fragmentation losses*[5] are reasonably small, and these resource partitioning mechanisms can substantially reduce the computational overhead associated with running admission tests. Also, it is critical that resource partitioning mechanisms work well with the other components of the multi-party real-time

[4]In Chapter 3 (as well as Chapter 4), we duplicate, for ease of reading, the simulation scenario information that we first present in Chapter 2.

[5]These losses are defined and described in Chapter 2.

communication service; we conclude Chapter 3 with a discussion of these interactions.

Chapter 4 presents advance reservations. The ability to reserve real-time connections in advance is essential in distributed multi-party applications using a network which controls admissions to provide good quality of service. We first discuss the requirements of the clients of an advance reservation service, and a distributed design for such a service. It is interesting that in addition to providing a much-needed service to these applications, the advance reservation mechanisms also improve the network service with better planning (network dimensioning) and routing [47].

We evaluated the performance of our advance reservation mechanisms via simulations; the simulation results demonstrate the usefulness of the mechanisms that we designed. These simulations also provide useful data about the performance and some of the properties of these mechanisms. Again, it is critical that advance reservation mechanisms work well with the other components of the multi-party real-time communication service; we conclude Chapter 4 with a discussion of these interactions.

In Chapters 2, 3, and 4, we present resource sharing, resource partitioning, and advance reservations, the three cornerstones of our multi-party real-time communication research. In Chapter 5, we describe how these components fit together to provide multi-party real-time communication service in the Tenet Scheme 2 (and the associated protocols, the Tenet Suite 2). We first discuss the design goals for the signaling protocol (RCAP) and describe how these design goals led to our design decisions. We then describe the object-oriented design of RCAP software and illustrate these interactions with a simple connection establishment example, along with some preliminary measurements on our prototype implementation; this implementation includes support for resource sharing, resource partitioning, and advance reservations.

Supporting multi-party applications also requires two key changes in RTIP, the real-time data delivery protocol: for multicasting and for resource sharing. We describe these changes also in Chapter 5, and we conclude that chapter with a discussion of some of the interactions between resource partitioning and advance reservations in our implementation.

Chapter 6 reviews related work by other researchers; we contrast our approach and research with that done by the designers of RSVP and ST2+. We first describe the differences in the design goals for each project, and we then describe how these different objectives led to the differences in the selected mechanisms. As ST2+ and RSVP are currently under development, we primarily restrict this discussion to the current proposals, though, for a few selected topics, we will also describe the other alternatives under consideration.

Chapter 7 summarizes the dissertation by discussing our contributions and the weaknesses of our approach. We also outline current research trends as well as the directions in which this work can be extended.

# Chapter 2

# Resource sharing

Many classes of applications, including distributed multimedia group communication [84] and traditional distributed processing, require or benefit from a network communication service that provides well-defined performance guarantees. A number of protocols and schemes have been proposed to provide real-time communication services [18, 49]. These schemes are usually connection-oriented, in that they allocate network resources (e.g., bandwidth, buffers and so on) along the path data packets will travel.

Traditional real-time network systems (e.g., [44]) may over-allocate resources for two reasons: (1) they allocate resources based on a worst-case prediction of the actual traffic; and (2) they treat traffic on different connections independently when determining resource requirements. One technique to improve utilization (and hence the connection acceptance rate) is to measure the actual traffic parameters of individual connections and to modify the amounts of resources allocated to the connection dynamically [94]. Another technique uses performance measurements over aggregations of connections to predict future performance [18]. The first approach still over-estimates aggregate resource requirements, since it fails to capture some important relationships between connections (e.g., in a conference, usually only one speaker is active at a time). The second approach will indirectly capture these relationships, but it fails to provide protection against unrelated traffic[1], and depends on the assumption that current behavior adequately predicts future behavior for all connections in the aggregate. This assumption may not be valid when a single connection can have a significant effect on the performance of other connections, e.g., over low-capacity links. In addition, when measurements are not available (e.g., when a connection is first established), this approach must fall back to the use of independent traffic characterizations, as in the first approach.

In this chapter, we present *resource sharing* as a middle ground, by which *related* connections can *share* resource allocations in a controlled manner, so that network utilization is improved and performance guarantees of established connections are achieved. Resource sharing differs from techniques that rely on *statistical multiplexing* of (unrelated) network traffic, in that the network client specifies how traffic from related connections may be

[1]If traffic measurements over aggregations are used to predict the future traffic, the performance seen by a channel will suffer if other channels (of the aggregate) increase their data rates in excess of the network prediction.

multiplexed. As long as the aggregate traffic of these related connections does not exceed this specification, the network service guarantees that well-defined performance bounds will be met for individual channels. As the client specifies the related connections and their aggregate traffic, all sharing is completely client-controlled. Most importantly, performance guarantees are *not* dependent on the behavior of unrelated network traffic. Resource sharing is an important technique to provide well-defined performance guarantees for most large-scale, multi-party communication paradigms.

In this chapter, we first motivate resource sharing with a simple example in Section 2.1. In Section 2.2, we describe fully-distributed mechanisms for doing resource sharing with real-time guarantees in a general internetworking environment; we do so in the context of the implementation of such mechanisms in the next generation of the Tenet real-time protocols. We illustrate these mechanisms with a simple example in Section 2.3. In Sections 2.4 and 2.5, respectively, we present analysis and simulation-based evaluations which show that resource sharing leads to a large gain in the connection acceptance rate, and a significant reduction in the computational overhead associated with admission control. Resource sharing is a key component of our multi-party real-time communication system; of particular interest are the interactions of resource sharing mechanisms with the other components of our system. We discuss these interactions in Section 2.6. We conclude this chapter with a brief summary in Section 2.7.

## 2.1 Motivation for resource sharing

In this section, we motivate resource sharing with a simple example. We present a simplified tele-conferencing scenario to motivate the need for resource sharing. Consider the simple conference scenario presented in Figure 2.1, where a conference is set up among $A$, $B$ and $C$ ($X$ is an intermediate node or router). Only the two multicast channels from A and from B are shown in Figure 2.1.

Due to the cooperative nature of the conference, it is reasonable to require that only one person speaks at any time. Indeed, in an orderly meeting only one person speaks at any time; two persons speak simultaneously only when they try to *get the floor*; clearly, this situation lasts for but a short period of time, and it should be acceptable if the performance degrades during that time period. For simplicity, we restrict the sharing in this example to audio streams. Similar sharing can be expected in video channels as well, either because the senders refrain from sending video when they do not have the floor or if the video application enforces mutual exclusion, like the dynamic window switching mechanism of *vic* [78, 79].

Consider the link $X$-$C$; the two multicast channels (from A and from B) can share the resources on the link $X$-$C$. Without resource sharing, the network will make independent reservations for the two channels, and wastefully over-allocate resources on the link $X$-$C$ by 100%. Under resource sharing, the client will inform the network that the two channels are both part of the same conference, and that the aggregate traffic on the channels will not exceed the traffic due to one source. The network can use this information to limit the resource reservations on the link $X$-$C$.

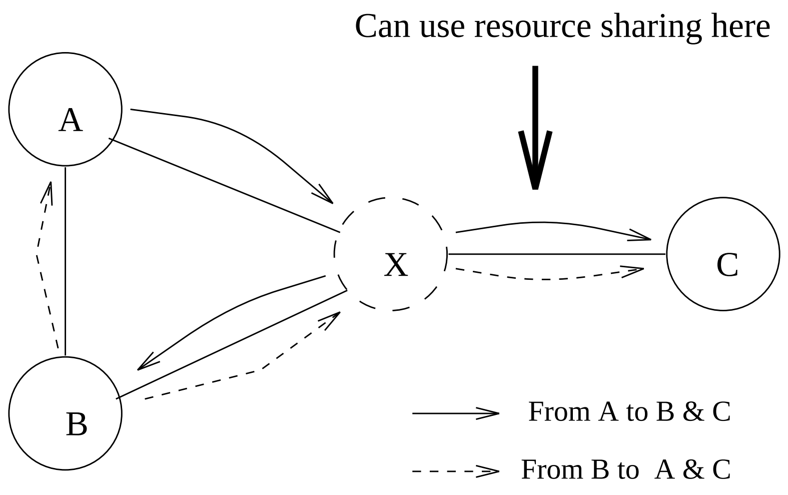

Figure 2.1: The 3-user Conference Example

This example illustrates a scenario where only one source is active at any time in the conference. In general, we can have up to $n$ concurrently active sources. We define the *maximum concurrency* for a conference as the maximum number of concurrently active sources; in the example above, the maximum concurrency is equal to one.

While the above example illustrates the need for resource sharing in a simple conference scenario, it should be noted that resource sharing is equally useful in other real-time multi-party scenarios such as panel discussions, distributed seminars and so on. For these multi-party applications, the maximum concurrency is usually smaller than the number of senders and, most significantly, *does not increase with the number of participants.* In such cases, resource sharing leads to more efficient use of network resources. In fact, since in most cases we expect the maximum concurrency to remain fairly small even when the number of participants increases dramatically, resource sharing enables better scalability for large conferences; the gains increase with the size of the conference.

## 2.2 Mechanisms for resource sharing

The key motivation for resource sharing is to exploit the known behavior of related channels in order to reduce the aggregate network resources allocated to these channels. To be attractive, resource sharing must give network clients the same performance guarantees that they would have received without resource sharing. The mechanisms we have devised are completely distributed; hence, they do not restrict the scalability of communication, and are robust in the presence of node and link failures. Indeed, simulation results ( presented in Section 2.5 as well as in [61],[62]) show that resource sharing improves the scalability of communication. Three types of mechanisms are required:

- *Client-service interface:* The network client must inform the network of sharing relationships between channels. This interface defines the contractual agreement between the client and the network.

- *Admission control tests:* The network admission control tests may use the information supplied by the client to perform local admission control tests on a group of channels, rather than on each channel individually.
- *Protection:* The network must ensure that network resources consumed by the channels in a group do not exceed the resource allocation of the group.

### 2.2.1 Client-service interface

To allow the network to share resource allocations between related channels, the client must specify three kinds of information:

- *A list of related channels that may share resources.* In [63], we defined the *channel groups* abstraction to enable clients to specify inter-channel relationships to the network. To specify a list of channels that may share resources, we define a channel group with a *resource sharing* relationship. Individual channels then join the channel group to share resources with other member channels[63].
- *Resource requirements for each group.* Our assumption is that at any given time the actual resources required by all group members will not exceed the resources allocated if the channels were treated independently. To benefit from this situation, the client must specify the maximum aggregate resource requirements for the channel group. Two approaches can be used:

    1. The client can specify directly the maximum aggregate resource requirements of the group of channels, or
    2. The client can specify the maximum concurrency between channels, and the network can compute a maximum resource requirement for the aggregate along each link.

  We have chosen the first approach, because, in the case where the maximum concurrency between channels is greater than one (e.g., when several video channels are displayed during a seminar), the client may specify a resource requirement for the combined streams that takes into account gains from statistical multiplexing between *related* channels. In the second alternative, the network does not know how traffic on separate channels may combine, and thus must treat channels independently of one another.
- *When the group requirements should be used.* In the case where the maximum concurrency is greater than one, the group requirements may be significantly greater than the resources required by any individual channel. Therefore, the client must indicate

to the network when to use the group specification rather than the individual ones. We take the simplest approach and specify a *sharing threshold* that corresponds to the maximum concurrency of the group. When the number of member channels on a link equals or exceeds the threshold, the group specification (described above) should be used. Before that time, resources are reserved for each channel independently of the others in the group. The alternative approach would be for the network to compare resource allocations for individual channels with that for the group aggregate and thus make this decision without the client explicitly specifying a *sharing threshold*. However, the network code is greatly simplified when the client explicitly specifies the sharing threshold; the network code can ignore this information if it can compare individual and aggregate resource allocations.

### 2.2.2 Admission control

The admission control tests determine if a new channel can be admitted without potentially violating the guarantees given to established channels. As described in [7], the Tenet protocols utilize a fully-distributed technique for connection establishment and admission control. The modifications to support resource sharing maintain this fully-distributed property. The key change is that the group resource allocation is used in admission control tests instead of the individual (per-channel) allocations when the number of member channels at a server equals or exceeds the threshold. After the sharing threshold has been reached, *no admission tests need be performed to admit additional member channels.*

Table 2.1: Link bandwidth allocation for a sharing group vs. number of channels

| No. of channels | Threshold | Sum of channel specs. (Mbps) | Group spec. (Mbps) | Reservation (Mbps) |
|---|---|---|---|---|
| 1 | 3 | 1.5 | 4.0 | 1.5 |
| 2 | 3 | 3.0 | 4.0 | 3.0 |
| **3** | **3** | **4.5** | **4.0** | **4.0** |
| 4 | **3** | **6.0** | **4.0** | **4.0** |
| ... | ... | ... | ... | ... |
| **n** | **3** | **n * 1.5** | **4.0** | **4.0** |

As noted in Section 2.2.1, we have decided to allocate resources according to the individual specifications until the number of channels using the server reaches the threshold, at which time we will switch to the group specification. Table 2.1 shows an example of the bandwidth allocated to channels from a sharing group at an arbitrary server[2]. In this example, we assume that each channel requires 1.5 Mbps bandwidth, the sharing threshold is 3, and the maximum bandwidth required by any three channels is 4.0 Mbps (because of the manner in which the streams are known to multiplex). The table shows the total bandwidth allocation for the channels as the number of member channels established through the server increases. When the first channel is established, the threshold has not been reached,

[2]A server is any network node or link where resources like buffers and/or delay may be allocated.

so 1.5 Mbps is reserved according to the individual specification of the channel. Likewise, the second channel reserves 1.5 Mbps according to its individual specification. The third channel, however, reaches the threshold, so the allocation is changed to the group specification. As can be seen in Table 2.1, once the threshold has been reached, no tests have to be performed for new channels in the group. This simple example shows how resource sharing improves scalability for large conferences.

To simplify changing from individual specifications to the group specification, all channels are given the same local delay bound during establishment, even when the scheduling discipline would allow us to give separate delay bounds per channel (such as with Earliest Due Date scheduling [75]). If a given channel requires a tighter delay bound than the bound given to group members, that channel should be established separately from the group at one or more servers. For example, if the Rate-Controlled Static Priority (RCSP) scheduling algorithm [125] is used, a channel is assigned a static priority and admission control algorithms ensure that a pre-specified delay bound is met for each priority level. To implement resource sharing, we substitute the group specification for the individual channel specifications in running the admission tests and delay bound computations. In our implementation of resource sharing for servers using RCSP scheduling, link bandwidth is allocated to the entire channel group according to the group specification. Therefore, all channels of the group receive the same bound on queueing delay.

### 2.2.3 Protection

In order to provide guarantees, we must ensure that each channel can use the resources that have been allocated for it. The rate control and scheduling routines do the *policing* that provides this protection. The Tenet protocols, for instance, protect the resource allocations of real-time channels (i.e., channels that make resource reservations) by giving them higher priority than non real-time channels, and by ensuring that no real-time channel exceeds its resource allocation. The main mechanism for policing real-time channels is rate control, i.e. the network ensures that the traffic for a channel does not exceed its specification[3]. Scheduling priority is protected automatically by the scheduling algorithm, and buffer space allocations are protected by allocating buffers to real-time channels.

To provide protection in the presence of resource sharing, we must provide the same level of policing on group aggregate traffic. To meet this requirement, we allocate resources to the group: when the group specification is in effect, all channels in a sharing group share common resources. Rate control and scheduling are performed by treating all traffic from member channels as belonging to a "super channel" that must obey the group specification. Only one addition to the normal, per-channel versions of these mechanisms is required to support resource sharing: when the group threshold has been reached in a server, rate control and scheduling are performed using the group allocation rather than

[3]This is not strictly true. In case of EDD, we can allow traffic for a channel to exceed its specification, without violating other channels' performance guarantees, by extending the deadlines. Similar results can be shown for fair-queueing-based scheduling disciplines. We only refer to strict rate-control-based policing for simplicity of discussion; the techniques described here are equally applicable to the approaches that allow more traffic to go through.

the allocation for the individual channel. To implement this change, we introduced an indirection from the channel table to the resource allocation records used by the rate control and scheduling algorithms. The algorithms themselves do not change. The organization is shown in Figure 2.2.

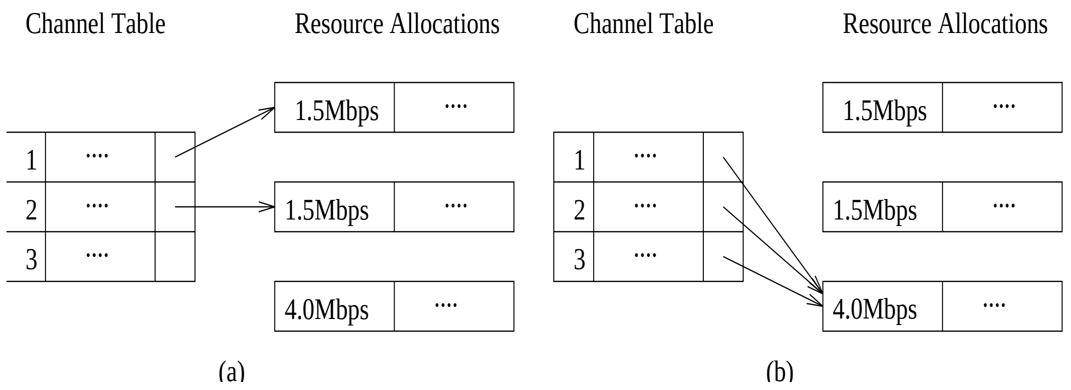

Figure 2.2: Implementation of rate control to support resource sharing using (a) individual allocations; (b) group allocation.

## 2.3 Example

In this section, we illustrate, with the help of a simple example, the various aspects of resource sharing described in the previous section. For simplicity, we consider a simple conference in a small tree-topology network shown in Figure 2.3.

We assume each conference participant ($P1, P2, ...P6$) is the source for one video channel with the bandwidth of 1.5Mbps to all other participants; the maximum aggregate resource requirement for all channels is 4.0 Mbps; and the sharing threshold is 3. In this case, the required resource allocations are as given in Table 2.1. For simplicity, we also assume that all requests are made by a single conference organizer, who could exist anywhere in the network.

To set up resource sharing, the conference organizer performs the following actions:

- Request the network to create a Multicast Group[4] ($MG1$), and to add each participant to the Multicast Group.
- Request the network to create one channel from each participant to $MG1$; all channels are 1.5 Mbps.

[4]This multicast group is the real-time analog of the IP HostGroup abstraction[27, 28], in that, in such a group, we also associate real-time performance requirements with the destinations interested in listening to a "session". In other papers (as well as later in Chapter 5), we have referred to these multicast groups as "Target Sets".

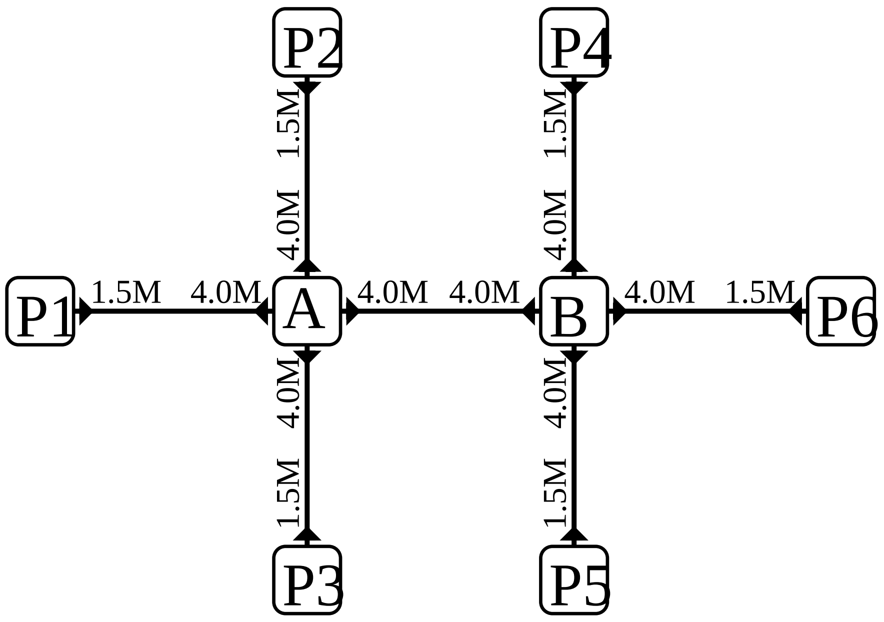

Figure 2.3: Resource allocations for the 6-user conference

- Request the network to create a resource sharing group (say $RSG1$) with aggregate resource allocation 4.0 Mbps and threshold of 3, and add the channels to the group.
- Request the network to establish these channels.

Below we describe the admission control process for link $B \rightarrow P6$. During establishment, similar computations take place at other links. At this server, five admission control requests arrived for this conference—one for each channel from participants $\{P1, ...P5\}$ to $P6$. The admission control actions for each of the establishment requests at this server are listed below:

- **First request arrives**: The admission control system notes that the request is for the group $RSG1$, and that the threshold has not been locally reached. Therefore, 1.5 Mbps is reserved for the channel according to the individual channel specification.
- **Second request arrives**: The admission control system again notes that the request was for the group $RSG1$, and that the threshold has not been reached. Therefore, another 1.5 Mbps is reserved for the channel according to the individual channel specification.
- **Third request arrives**: Since the threshold has now been reached, the allocation is changed to the group specification of 4.0 Mbps. The protection and traffic policing is as shown in Figure 2.2.

- **Fourth and fifth requests**: Since the threshold has been reached, no tests have to be performed and the requests are immediately accepted.

## 2.4 Analysis

For an analytical evaluation of resource sharing benefits, we introduce the *allocation gain* metric. For a set of connections over a set of links, we define allocation gain as the ratio of the allocation of a given resource required without resource sharing, to the allocation of that resource required when resource sharing is used. For example, an allocation gain of 4 means that, under resource sharing, $\frac{1}{4}$ as many resources are required as without resource sharing. We chose allocation gain as the metric for our analysis because, in our fully-distributed mechanisms, the resource sharing benefits accrue on a per-link basis, and it is difficult to compute from them the gains in overall channel acceptance. In the next section, we will use a different metric called *acceptance gain* for evaluating the resource sharing benefits in the simulation experiments.

A simple example can show that, under some conditions (very dense networks, adversarial routing, and so on), resource sharing will not provide any useful gains in resource allocation. Consider a complete network, i.e., direct links connecting all pair of nodes, and a routing algorithm that ensures that data packets go directly from the source to the destinations, with no intermediate nodes. If only one sender (of a resource sharing group of senders) resides on any network node, no link will carry data belonging to more than one channel; consequently, we cannot obtain any savings by introducing resource sharing in this case.

However, we do expect that, in sparser computer networks, we will obtain significant savings with resource sharing. We present two analyses to demonstrate these savings: the first analysis is for sparse networks, while in the second analysis we assume that, for each conference, the network routing function creates a single undirected tree and returns routes along that tree for all connections that constitute that conference. We present useful results for resource sharing gains under these sets of assumptions.

### 2.4.1 Sparse network

Wide-area networks (WANs) tend to be rather sparse; for example, the NSFNET backbone WAN has 32 nodes and only 35 links[5]. In this analysis, we obtain lower bounds on resource sharing gains for networks with small cut-sets of links. It should be noted that sparse networks usually have small cut-sets of links. For the purpose of this analysis, we assume that the conference participants are homogeneous and are uniformly and independently distributed among the network nodes. To keep the analysis general, we make no assumptions whatsoever about the routing algorithms.

[5]The NSFNet backbone has been replaced now by a group of “commercial” networks.

**Lemma 2.1** *Consider a network in which channels always traverse at least one link of a set of links $L$, with $|L| = l$. If a conference needs $m$ channels, then a lower bound, $g$, on the resource sharing allocation gain is given by*
$g = \frac{m}{2tl}$,
*where $t$ is the sharing threshold specified by the client.*

The lemma follows from a simple counting argument: without resource sharing, at least one reservation for each resource must be made over the links of the set $L$ for each of the $m$ channels, for a minimum of $m$ reservations over the set $L$. With resource sharing, at most $t$ reservations must be made (in each direction) for each link of set $L$. Since a link has only 2 directions, a maximum of $2tl$ reservations are required.

For a conference with 40 channels, $t = 2$, and $l = 5$, the allocation gain $g$ always exceeds (or equals) 2.

**Lemma 2.2** *Consider a network with a cut-set $L$ of links that divide the network into a set of subnetworks, such that no subnetwork contains more than a fraction $f$ of the nodes of the original network. If a channel has $n$ destinations uniformly and independently distributed among the network nodes, then the probability that the channel traverses at least one of the links of the cut-set $L$ is given by*
$prob \geq 1 - f^n$.

A channel must traverse a link of the cut-set $L$ *unless* all destinations reside in the same subnetwork as the source. Assuming a uniform, independent distribution of destinations, the probability that a single destination resides in the same subnetwork as the source is at most $f$. Thus, the probability that all $n$ destinations reside in the same subnetwork as the source is at most $f^n$.

As the number of destinations increases, the probability *prob* rapidly approaches 1; for example, with $f = 0.75$, $n = 16$, probability *prob* is greater than 0.99.

**Theorem 2.1** *Consider a network with a small link-set $L$ with $|L| = l$, where $r$ is the probability that a channel traverses the link-set $L$. If a conference has $m$ channels and the sharing threshold $t$, the following relation holds:*
$\log(\frac{1}{1-p}) = \frac{mr}{2}(1 - \frac{2tlg}{mr})^2$,
*where $p$ is the probability that the allocation gain is at least $g$.*

We use the Chernoff bound on a sum of independent Bernoulli trials (first described in [14]) to obtain lower bounds on resource allocation gains that result from resource sharing. This Chernoff bound states that, for independent Bernoulli trials with $P[X_i = 1] = p_i, p_i \in (0, 1)$, and random variable $X$, where $X = \sum_{i=1}^{n} X_i$, and $\mu = \sum_{i=1}^{n} p_i > 0$,

$$P[X < (1 - \delta)\mu] < exp(-\mu\delta^2/2),$$

where $\mu$ is the expected value of $X$. We rearrange the expression to

$$P[X \geq x : x = (1 - \delta)\mu] \geq 1 - exp(-\mu\delta^2/2).$$

Let $X$ be a random variable representing the number of channels (out of the $m$ channels that comprise the conference) that traverse the link-set $L$. Here, $\mu = E[X] = mr$. From Lemma 2.1, the gain factor will exceed $g$ iff $X > x$, where $x = 2tlg$. Setting $x = (1-\delta)\mu$ yields $\delta = (1 - \frac{2tlg}{\mu})$. Substituting $\mu$ and $\delta$ into the previous expression,

$$P[X \geq x] \geq 1 - exp\left[-mr(1 - \frac{2tlg}{mr})^2/2\right].$$

Let $p$ denote the lower bound on the probability $P[X \geq 2tlg]$; thus,

$$p = 1 - exp\left[\frac{-mr}{2}(1 - \frac{2tlg}{mr})^2\right].$$

Rearranging the above expression, we obtain:

$$\log(\frac{1}{1-p}) = \frac{mr}{2}(1 - \frac{2tlg}{mr})^2.$$

This theorem shows the relationship between $m$,$r$,$g$,$t$,$l$, and $p$; for instance, $p$ decreases as $g$ increases if the other factors are kept constant. As an example, the probability $p$ is greater than 0.95 for $g = 2$, with $m = 50$, $r = 0.99$, $l = 4$ and $t = 2$.

### 2.4.2 Tree-based routing

In this section, the analysis shows the strong relationship between routing and resource sharing gains. We adopt a simple routing strategy that attempts to increase the sharing gains by selecting the same *undirected* routing tree for all channels that belong to a particular conference. One way of looking at this tree selection process is that the routing system selects a spanning tree $T$ for the network. Figure 2.4 shows a simple network in which, out of the set of network links (thin lines), the routing system has selected a spanning tree T (the links in $T$ are shown with thicker lines). Then for every channel, the appropriate *directed* subtree $t$, of $T$, that connects all destinations to the source is used. This behavior is exhibited by many current routing algorithms, including Core-Based Trees [1]. In this analysis, we relax the constraints that we previously imposed on the network topology and on the distribution of destinations among the network nodes; thus, this analysis is applicable to arbitrary network and connection topologies.

To keep the analysis simple, we assume that, for a given conference, the routing algorithm selects the same forwarding tree, regardless of whether resource sharing is used. Admittedly, a routing algorithm for conferences that do not use resource sharing probably would not be tree-based, since such an algorithm would tend to cause congestion on the shared links. However, since our analysis merely calculates the resources *required* at each server rather than performing admission control (essentially assuming infinite resources are available) the assumption of tree-based routing does not adversely affect the analysis.

Consider the subset $L$ of links in the spanning tree $T$ that connect the destinations for the conference (in Figure 2.4, these links are shown with extra dashes). It is easy to see

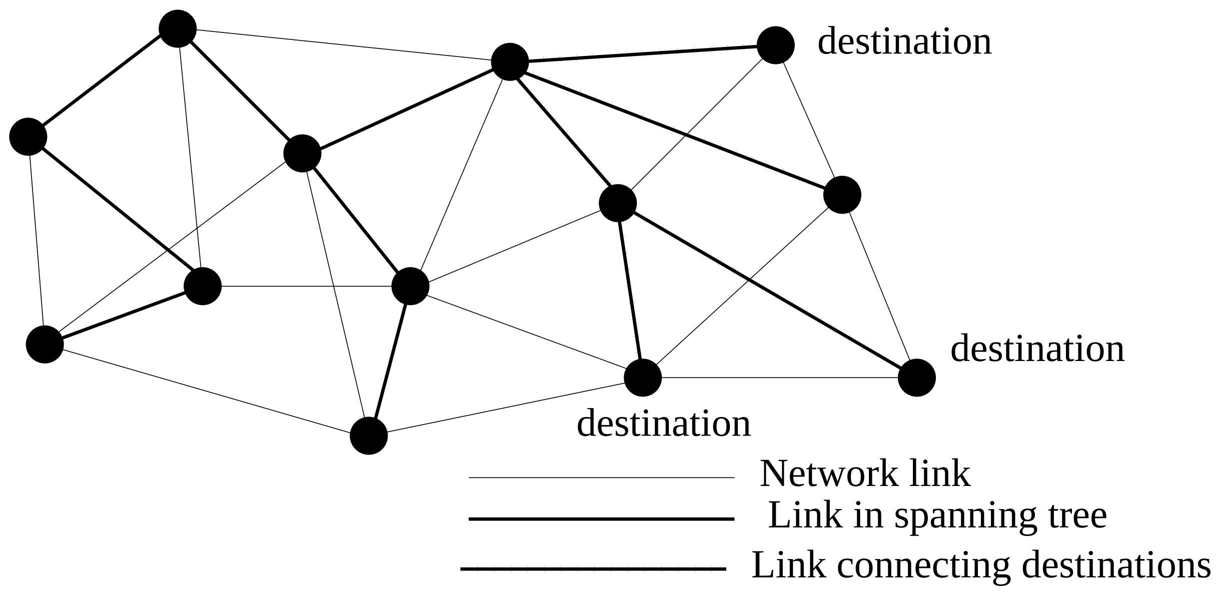

Figure 2.4: Network with tree-based routing

that every channel will traverse all links in $L$ exactly once in some direction. We can then obtain the following corollary of Lemma 2.1.

**Corollary 2.1** *Consider any link $l$ that belongs to the set of links $L$ which connects the destinations of the conference as described above. If a conference has $m$ channels and resource sharing threshold $t$, a lower bound $g$ on the allocation gain for each link $i$ is given by*

$$g = \tfrac{m}{2t}.$$

For a conference with 40 channels with a sharing threshold of 2, the allocation gain is at least 10.

Consider a link $i$ of the spanning tree $T$ that is used for routing channels for a given conference. The link $i$ divides the tree $T$ into two subtrees. Let the ratio of the numbers of nodes in the two subtree be $r_i$, with $r_i > 1$. Under the assumption that the destinations are uniformly and independently distributed among the nodes in the network, we obtain the following corollary of Lemma 2.2.

**Corollary 2.2** *If a channel has $n$ destinations, the probability $f$ that a randomly selected channel will traverse the link $l$ is given by*

$$\begin{aligned} f &= 1 - \left( \left( \frac{r_i}{1+r_i} \right)^{n+1} + \left( \frac{1}{1+r_i} \right)^{n+1} \right) \\ &= 1 - \frac{r_i^{n+1} + 1}{(1+r_i)^{n+1}} \end{aligned}$$

$$
\begin{aligned}
&> 1 - \left(\frac{r_i}{1+r_i}\right)^n = 1 - \left(1 - \frac{1}{1+r_i}\right)^n \\
&> 1 - e^{\frac{n}{1+r_i}}
\end{aligned}
$$

For example, $f > 0.999$ for $n > 10 * (1 + r_i)$.

The following corollary follows from Theorem 2.1 and Corollary 2.1.

**Corollary 2.3** *If a conference has $m$ channels and a sharing threshold $t$, the following relation holds for all links that belong to the routing tree for that conference:*

$\log(\frac{1}{1-p}) = \frac{mf}{2}(1 - \frac{2tg}{mf})^2$,

*where $p$ is the probability that the allocation gain is at least $g$, and $f$ is as given by Corollary 2 above.*

### 2.4.3 Examples

In this subsection, we illustrate the above analytically-derived bounds with some graphs; we have derived these graphs from Corollary 2.3.

For these graphs, we set the sharing threshold $tto2$, probability $p$ of Corollary 2.3 to 0.99 and the probability $fto0.999$ unless otherwise stated.

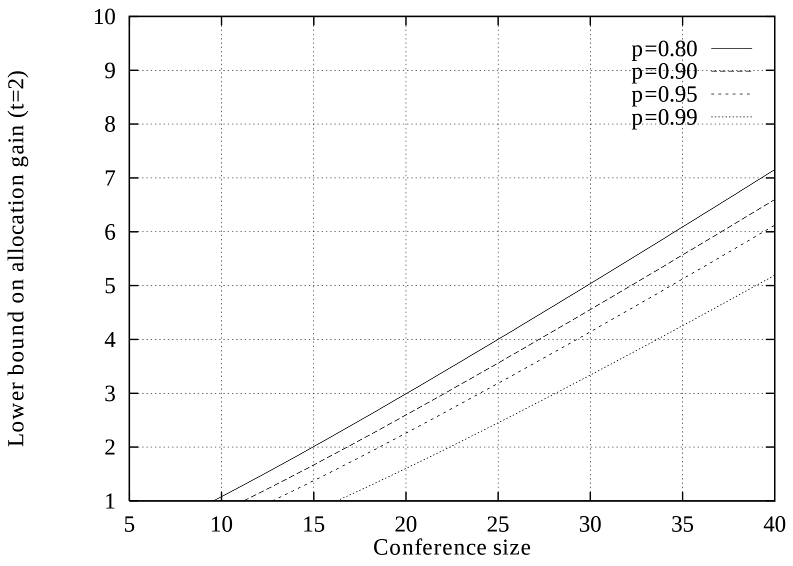

Figure 2.5: Analysis – Allocation gain for constant sharing threshold

In Figure 2.5, we fix the sharing threshold at 2, and obtain bounds for the allocation gains as we vary the conference size; we obtain curves for $p$ varying from 0.8 to 0.99, where $p$ is the probability that the allocation gain will exceed its bound.

As expected, at fixed probability $p$, the lower bound $g$ on the allocation gain increases almost linearly with the conference size; also, as the probability $p$ increases (thereby getting us closer to guaranteeing that the allocation gain will exceed the lower bound), the value of the lower bound $g$ decreases for the same value of conference size.

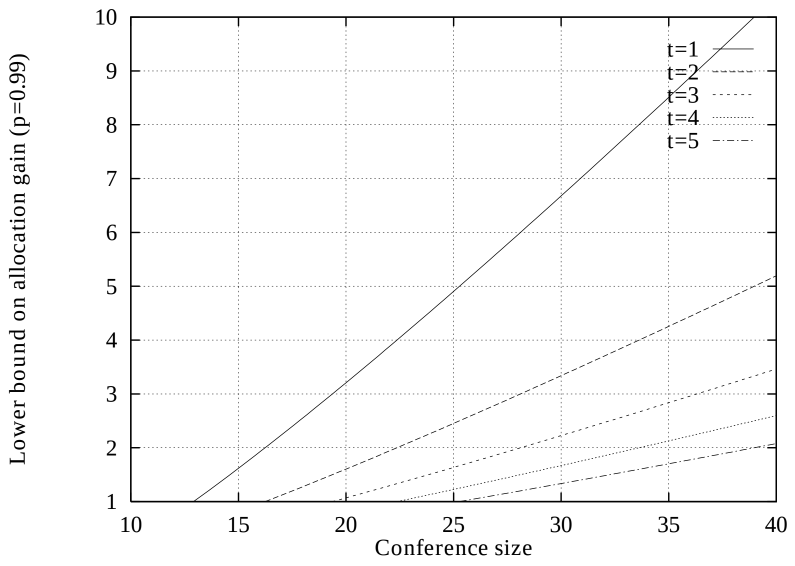

Figure 2.6: Analysis – Allocation gain for varying sharing threshold

In Figure 2.6, we fix the probability $p$ of exceeding the sharing threshold at 0.99, and obtain lower bounds on the allocation gain as we vary the conference size; we obtain curves for sharing threshold $t$ varying from one to five.

As expected, for a given sharing threshold, the allocation gain increases almost linearly with the conference size; also, as the sharing threshold increases, the lower bound on allocation gain goes down.

In Figure 2.7, we fix the sharing threshold at 2, and the probability of exceeding the allocation bound at 0.99 (i.e., 99% of the time, the actual allocation gain will exceed $g$). We obtain lower bounds on the allocation gain as we vary the conference size; we have plotted curves for $f$ varying from 0.8 to 0.999, where $f$ is the probability given by Corollary 2.

As expected, for fixed probability $f$, the lower bound on the allocation gain increases almost linearly with the conference size; also, as the probability $f$ increases (implying that the link is more likely to be traversed by most connections), the lower bound on the allocation gain increases.

## 2.5 Resource sharing simulations

In the previous section, we provided analytical bounds for the allocation gain due to resource sharing. In this section, we present the results of our simulations with resource sharing. We ran these simulations on *Galileo* [72], an object-oriented real-time network simulator. Our goal was to make the experiments realistic so that the results obtained can be

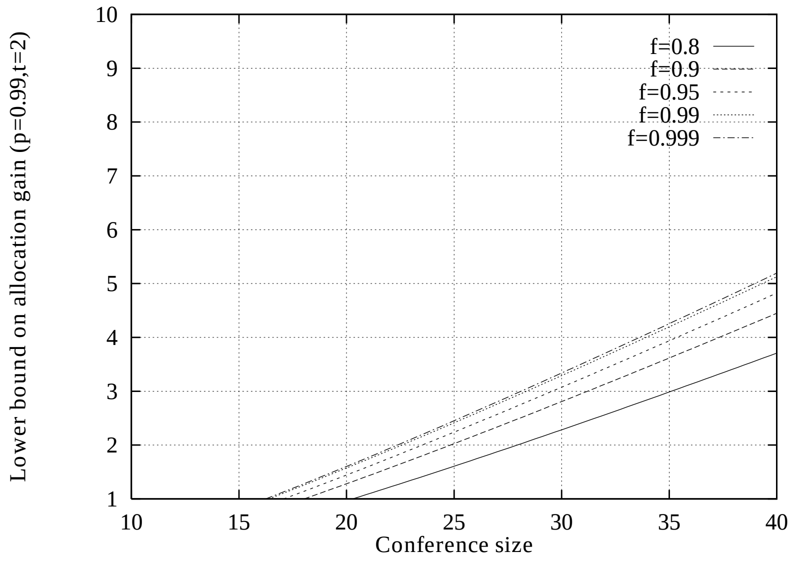

Figure 2.7: Analysis – Allocation gain for constant probability of exceeding the gain bound

confidently transposed to our resource sharing implementation in the Tenet Protocol Suite 2 [60]. For this, the network topologies that we used in the simulations are based on two real wide-area networks – the NSFNET backbone network, and XUNET [53], a high-speed ATM network that spans across North America from Bell Labs in New Jersey to Berkeley, California. We assumed the rate of each link to be 45 Mbps and the propagation delay along each link to be 5 ms. We also made the amount of buffer space in each server large enough that the bandwidth or processing power was the limiting resource in all scenarios.

### Simulation workload and evaluation metrics

In all the experiments, the sources and destinations for the conferences were uniformly and independently distributed among the network nodes. We ran the same simulations with and without resource sharing; in this section, we denote by RS the experiments with resource sharing and by non-RS the experiments without resource sharing.

To keep comparisons meaningful, we only considered a single type of traffic stream (i.e., a compressed video stream with a peak rate of 1 Mbps, 30 frames per second, and four data packets per frame), and destinations with identical performance requirements (end-to-end delay bound 400 ms); the average data rate did not matter, because the admission tests in our simulations only used peak-rate bandwidth allocation. Unless otherwise specified, the sharing threshold/maximum concurrency equaled one.

We performed many sets of experiments, each time varying one of three workload parameters: number of concurrent conferences, number of participants per conference, and sharing threshold. For each of these workloads, several trials have been run, and the results

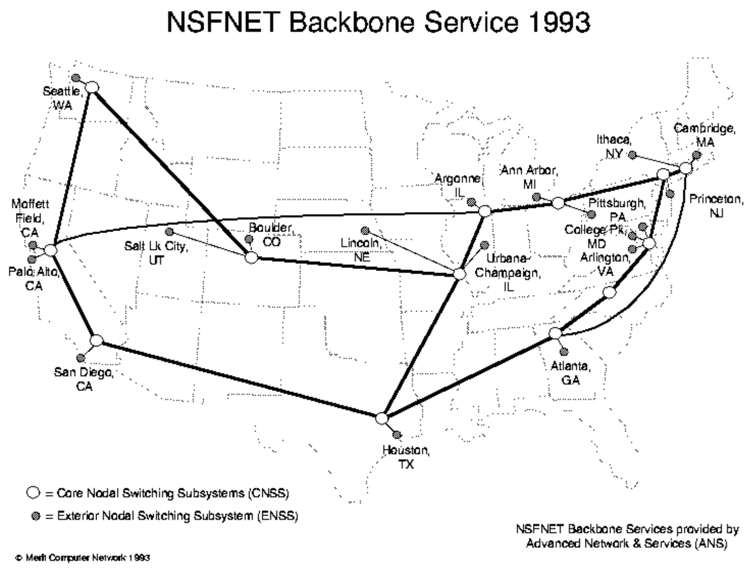

Figure 2.8: The NSFNET Network

obtained are averaged. In the first set, we varied the number of conferences, while keeping the number of participants in a single conference (*conference size*) fixed at 10. In the second set of experiments, we varied the conference size, and fixed the number of conferences at 50. We chose this workload to overload the system slightly, since the benefits of resource sharing are greatest under high network utilization.

The main metric for performance evaluation is:

$$\text{Destination Acceptance Rate} = \frac{\text{Number of destinations successfully established}}{\text{Number of destinations attempted}}$$

Here, *destinations* refer to the recipients in a multicast transmission. For the remainder of this paper, *acceptance rate* will refer to the *destination acceptance rate*, unless otherwise stated. We also define another metric, called *acceptance gain*, to be the ratio of acceptance rates with and without resource sharing respectively.

In addition, we are interested in the speed and computational cost of channel establishment, for which we use a different metric: the computational overhead associated with admission control. In the simulations, we use the admission control tests for the EDD scheduling discipline [49]. The time required to run these tests at a given node is proportional to $n$, the number of already accepted resource allocations at that node. The overall establishment overhead is computed as the sum of these $n$ (already accepted allocations) at each node along the route of the channel. This component of admission control computations increases as the real-time network load (number of accepted channels) goes up. Computational overhead for other activities (for example, for collecting the state information) does not change when this load increases (though it depends on the route selected). We therefore focus on this admission test computation overhead for performance

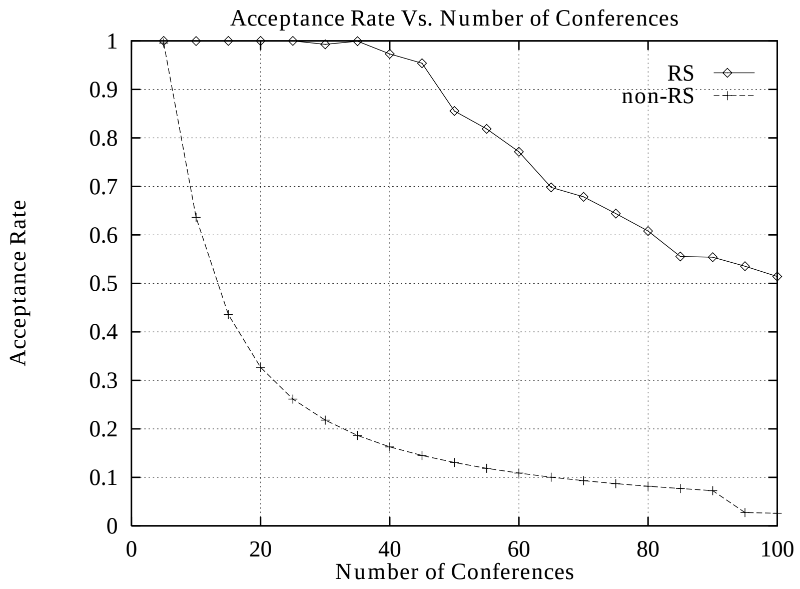

Figure 2.9: NSFNET – Acceptance Rate Vs. Number of Conferences

evaluation [6].

With resource sharing, since the established channels at a node can belong to sharing groups, and if the aggregate traffic specifications for such groups are being used, the number of individual resource allocations (buffer space, throughput, scheduling priority) $n$ can be much smaller than the number of channels. We can therefore expect a considerable reduction in the computational overhead when resource sharing is used.

### 2.5.1 NSFNET

In this set of experiments, we ran our simulations on the backbone of the NSFNET network; we only included the core nodes (CNSS) of the network shown in Figure 2.8.

#### Results

- Acceptance rate

  (i) Increasing the Number of Conferences
  With resource sharing, the acceptance rate is consistently higher than without resource sharing across a wide range of the number of conferences (from 5 to 100). From Figure 2.9, when the network is heavily-loaded (40 or more conferences), the destination acceptance rate with RS is at least 6 times higher than that for non-RS.

[6]In our comparison, we have ignored the extra overhead associated with accessing and maintaining resource sharing related information.

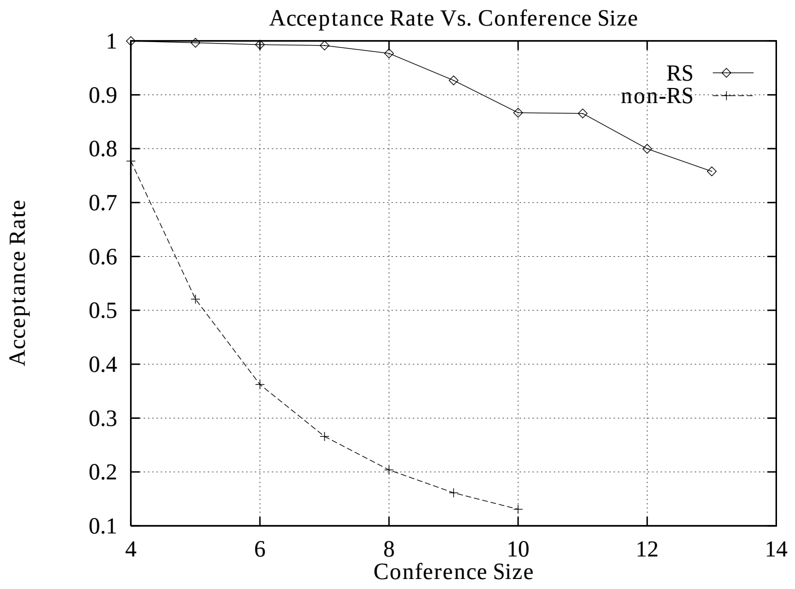

Figure 2.10: NSFNET – Acceptance Rate Vs. Conference Size

(ii) Increasing the Conference Size (Figure 2.10)
With RS, the acceptance rate remains close to 1 even when the size of conferences approaches the size of the network (in number of nodes), while the acceptance rate for non-RS degrades substantially as the conference size increases. This behavior is expected because, with resource sharing, the amount of network resources allocated to a conference is bounded by the sharing threshold, regardless of the size of the conference. This experiment illustrates the scalability and the importance of resource sharing for large multi-party applications.

(iii) Varying the Threshold
In Figure 2.11, we vary the sharing threshold in the sharing specification. In the graphs, RS$i$ refers to RS with a sharing threshold of $i$; RS1 denotes the case of sharing threshold = 1; RS2 denotes the case of sharing threshold = 2, and so on. A sharing threshold equal to the conference size (RS10 in Figure 2.11) amounts to turning resource sharing off. Figure 2.11 shows that, with a lower sharing threshold, the resource sharing gain is significantly higher, and that RS consistently outperforms non-RS.

- Computational Overhead

  According to Figure 2.12 and Figure 2.13, the admission control computation overhead for RS is always smaller than the overhead for non-RS. These results indicate that resource sharing does not add to the establishment overhead; indeed, resource sharing tends to *decrease* the establishment overhead. As described in Section 2.2, when the number of admitted channels in a sharing group reaches the sharing threshold at a server, the admission control mechanisms at *that server* will accept subsequent channels of the same group without performing any additional admission control tests.

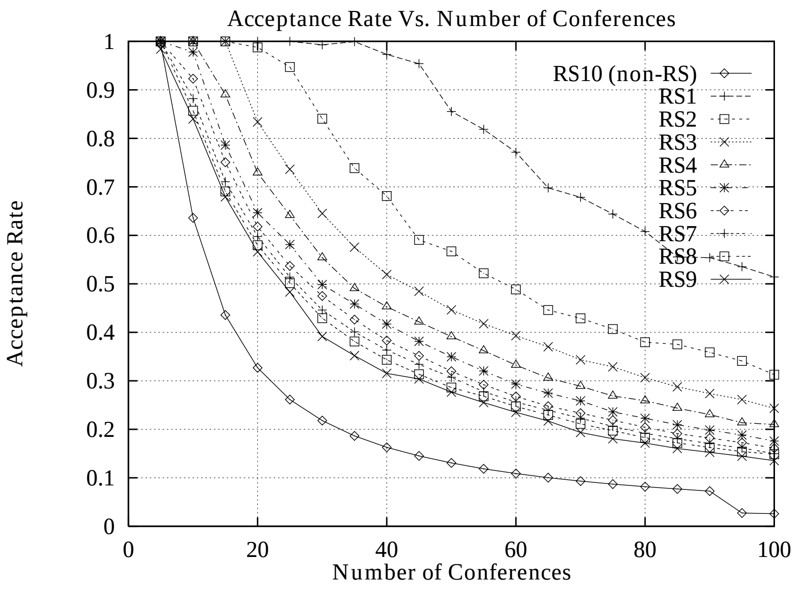

Figure 2.11: NSFNET – Acceptance Rate with Different Threshold Value

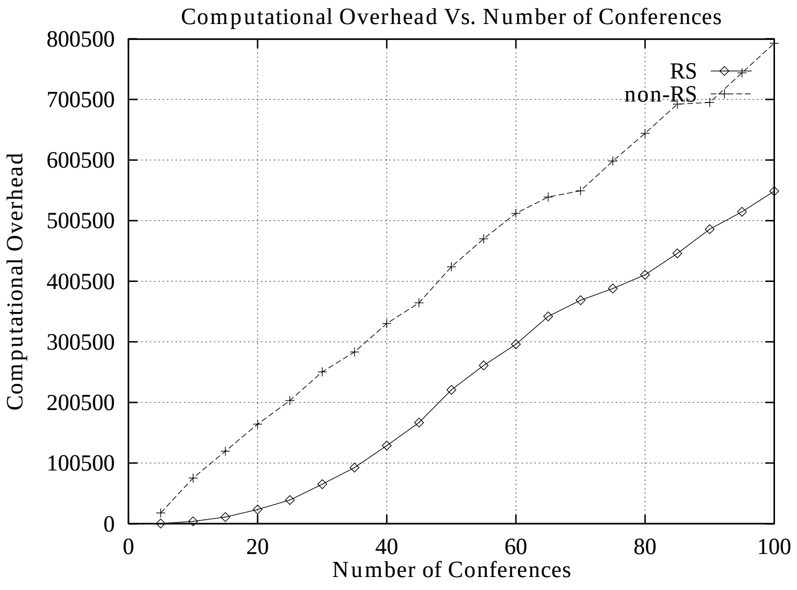

Figure 2.12: NSFNET – Computational Overhead Vs. Number of Conferences

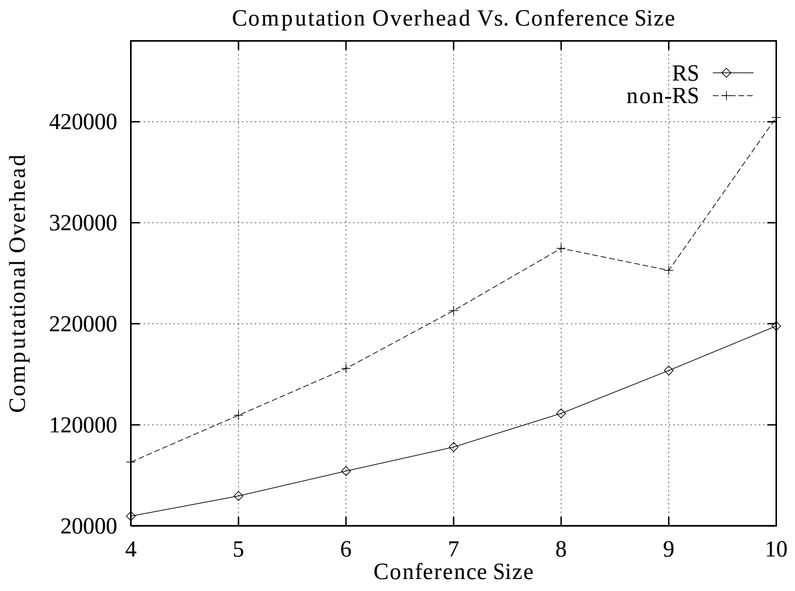

Figure 2.13: NSFNET – Computational Overhead Vs. Conference Size

### 2.5.2 Tree topology

We ran several simulation experiments with a tree-based topology (in which we added two extra hosts each at the four sites (Berkeley, Madison, Urbana, and Murray Hill) to the XUNET network topology of Figure 2.14).

#### Results

- Acceptance rate
  As in the previous section, we compare the acceptance rates obtained with and without resource sharing in a number of different experiments. The graphs (Figure 2.15 - 2.17) show how the system performance changes as we vary three parameters:

  (i) the number of conferences (Figure 2.15) ;

  (ii) the conference size (Figure 2.16) and

  (iii) the sharing threshold (Figure 2.17)

  The results are similar to the results that we obtained with the NSFNet topology. Resource sharing is shown to yield a higher acceptance rate.

- Computational Overhead
  Figures 2.18 and 2.19 show the computational overhead of RS and non-RS. The

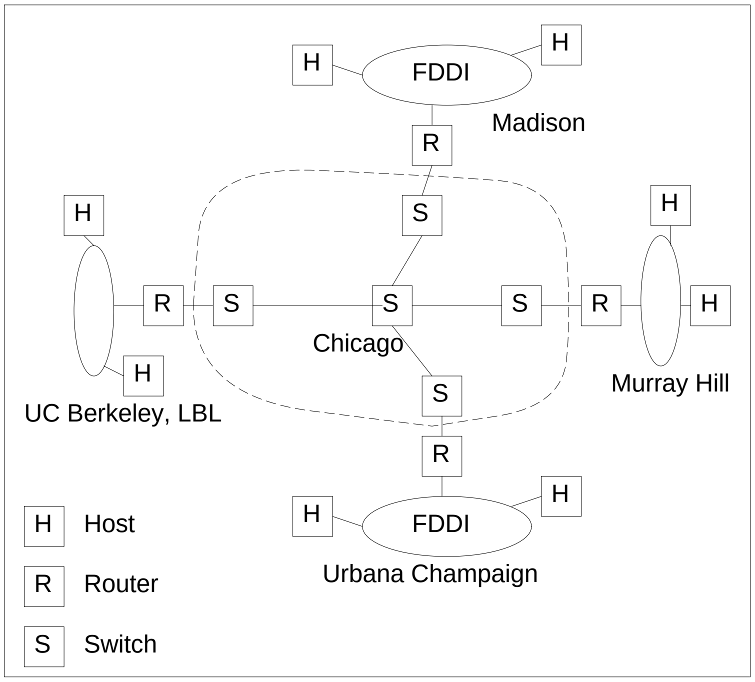

Figure 2.14: The XUNET Network

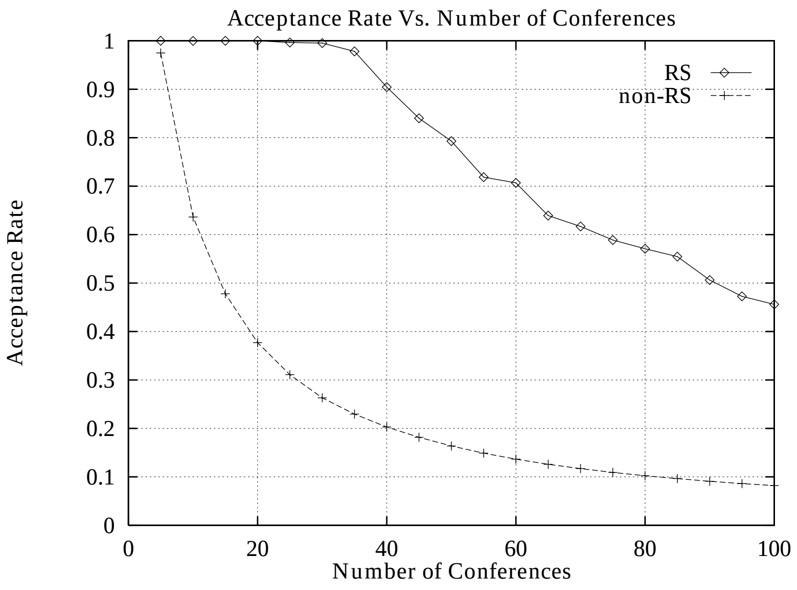

Figure 2.15: XUNET – Acceptance Rate Vs. Number of Conferences

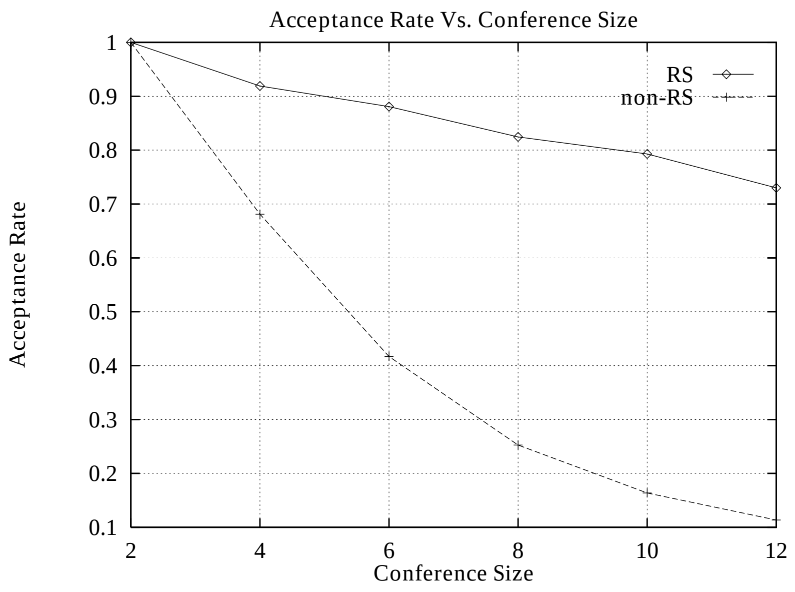

Figure 2.16: XUNET – Acceptance Rate Vs. Conference Size

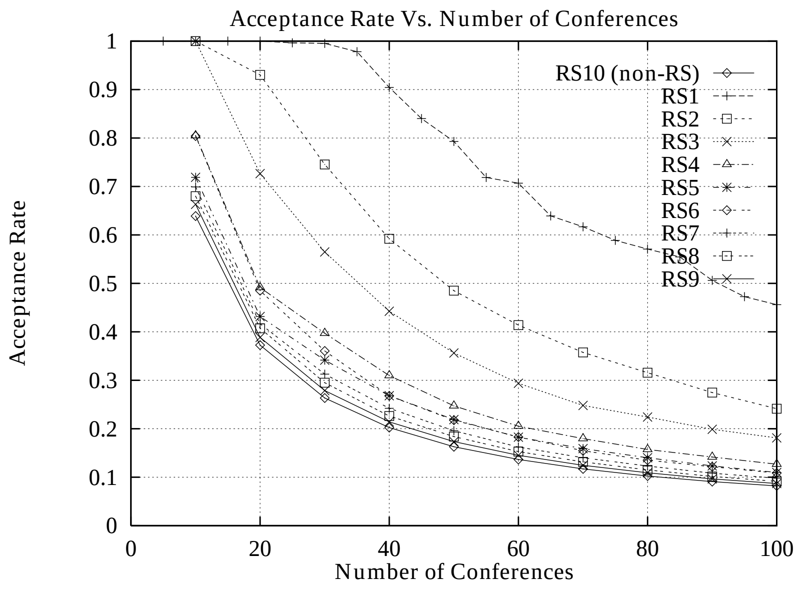

Figure 2.17: XUNET – Acceptance Rate with Different Threshold Value

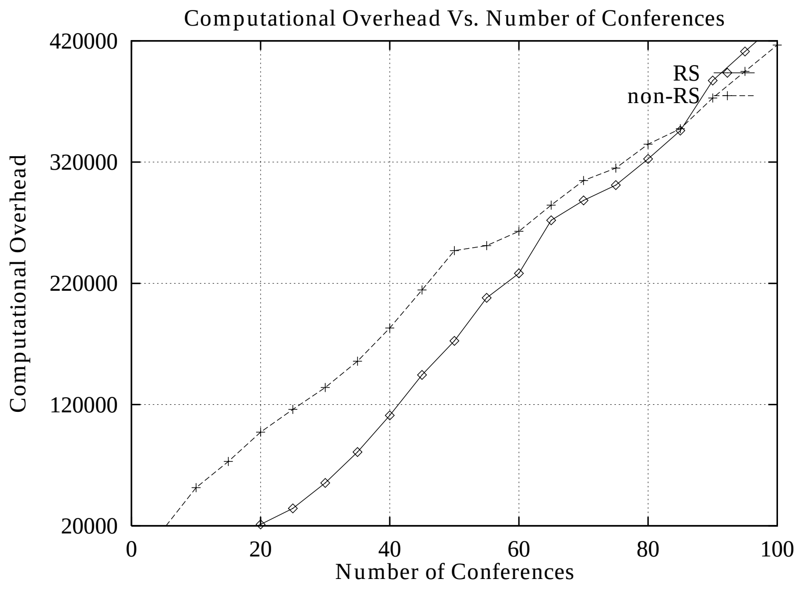

Figure 2.18: XUNET – Computational Overhead Vs. Number of Conferences

results are again very similar to those from the NSFNet in that RS in general reduces the computational overhead during channel establishment.

### Simulating heterogeneous traffic and performance parameters

We deliberately limited our simulation experiments to connections with homogeneous traffic specifications; this choice allowed us to define a suitable and sensible evaluation metric. With heterogeneous traffic, in our opinion, it is much more difficult to define such a metric[7]. For example, given that bandwidth is not the only resource to be considered, is a 1 Mbps channel worth 10 times as much as a 100 Kbps channel? Or is it worth a little less? A lot less? Similarly, if a client requests 100 ms as the bound on end-to-end delay and another client requests 200 ms, how do we compare their requests according to a common metric?

## 2.6 Interactions with other components

Resource sharing is only one component of our system for supporting multi-party real-time communication in computer networks; thus, it is very important that the resource sharing mechanisms not only work well, but that they work well with the other components

[7]The main evaluation metric – acceptance ratio – may also be criticized in the following manner: a multicast to 10 destinations should not be worth 10 times a unicast. However, we have not been able to come up with evaluation metrics that are simple and easy to obtain and to justify.

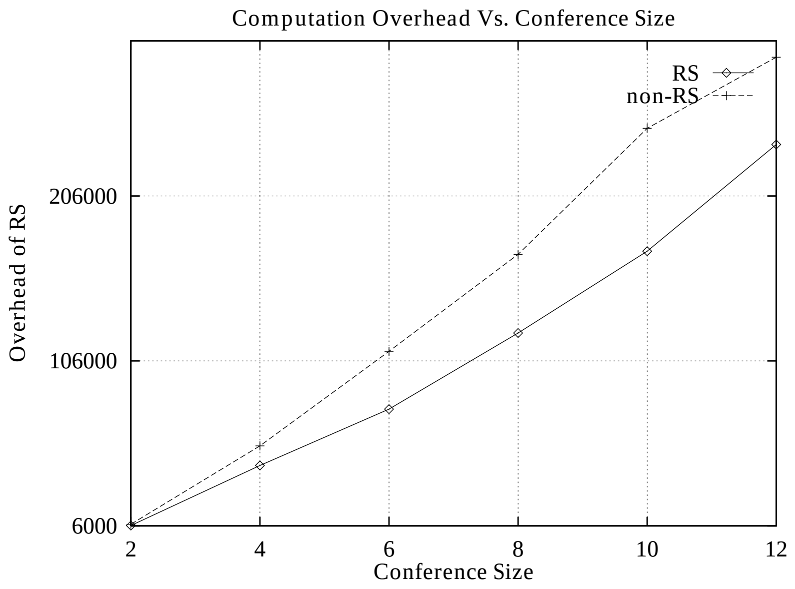

Figure 2.19: XUNET – Computational Overhead Vs. Conference Size

of our multi-party real-time communication system. In particular, resource sharing affects the following components: client-service interactions and interface, admission tests, traffic specification models, routing, access control, data forwarding and traffic policing, advance reservations, and resource partitioning. In Section 2.2, we discussed a few of these: we talked about how sharing affects client-service interactions, admission tests, and data forwarding and traffic policing.

In this section, we will discuss the interactions of resource sharing mechanisms with the following components of our real-time communication scheme: the local admission control mechanisms, the routing system, traffic specification models, and the mechanisms for supporting advance reservation of network resources. We will describe the interactions with resource partitioning in Chapter 3.

### 2.6.1 Resource sharing and local admission control

With resource sharing, the client promises that the aggregate traffic due to the related channels will remain within the client-specified bounds, and the network tries to use this information to reduce the resource allocation. An interesting issue is the interaction of this *globally specified* relationship with the admission control decisions that are made *locally* at the intermediate nodes (the local decision-making is due to the distributed nature of our resource sharing technique).

During channel establishment, the Real-Time Channel Administration Protocol (RCAP) [76] is responsible for admission control and resource reservation at each node. When the RCAP module at a certain node is reserving resources for a new channel, it allocates enough buffers for handling all potential delay jitter of the data packets from the previous node. This delay jitter bound depends on the resources allocated to the channel at the previous node (and the scheduling discipline followed at that node). Since channels that belong to the same sharing group may be allocated different amounts of various resources at the respective previous nodes, data from different member channels may arrive at a given node with different delay jitter bounds. Thus, adding a new channel for an already established group may result in additional buffer allocation at that node.

### 2.6.2 Resource sharing and traffic specification models

For obtaining performance guarantees, the network clients characterize the channel's traffic in terms of a *traffic specification model*; the characterization is used in running admission tests and in allocating resources. With resource sharing, the clients also need to characterize the aggregate traffic (for a group of resource sharing channels).

For characterizing traffic for individual channels, many models exist, including peak rate, $\sigma$-$\rho$, leaky-bucket, Xmin-Xavg-I-Smax, double bucket, and so on [5, 19, 20, 49, 73, 86, 89, 111, 116]. Some of these are not appropriate for specifying aggregate traffic for groups with two or more concurrently active senders; this is mainly because packets from different senders may arrive at the same time ("back-to-back") at some node in the network – in this case, we cannot directly use traffic models[8] that specify minimum inter-packet interval (e.g., Xmin-Xavg-I-Smax, or token bucket); we can use other models like $\sigma$-$\rho$, or double bucket.

### 2.6.3 Resource sharing and the routing system

The analysis and simulations described in Section 2.4 support the intuition that routing techniques affect the success of resource sharing. The routing algorithm can increase resource sharing gains in two ways:

- Routing of channels in the same sharing group along common subpaths.
- Routing channels so that they may have the same local delay bound along common subpaths.

The first effect is obvious: if the routing algorithm increases the overlap between multicast trees for related channels, a higher resource sharing can be achieved. The second effect is the result of our decision (or the requirement of some scheduling algorithms) to give all channels in a sharing group the same local delay bound. If the routing algorithm is aware of the local delay bounds given to member channels, it can take these into account. Unfortunately, most routing algorithms do not enhance the overlap of related multicast trees. In fact, adaptive routing algorithms may actually push channels of the same group away from each other to "less loaded" paths [77, 67, 39, 123]. The only way for the routing algorithm to enhance

[8]For example, with the Xmin-Xavg-I-Smax model, for reasonable values of Xavg, I, and Smax, if the Xmin value is set to zero (for packets arriving back-to-back), we have to reserve a lot of resources to serve the packets, thereby reducing most of the resource sharing gains.

resource sharing performance is to recognize channels as members of sharing groups, and to favor shared subpaths between related channels. The Tenet group has devised a routing algorithm that uses a heuristic to find a *min-cost* route that is likely to satisfy the delay requirements of all destinations [123]. They have also adapted this algorithm to maintain state information for each sharing group, both to enhance path overlap and to meet the delay restrictions specified above.

### 2.6.4 Resource sharing and advance reservations

We have also devised techniques for providing *advance reservations* for multi-party real-time communication [47]. In this service, the clients can make a request for network resources in advance of their use. A client may, for example, request a connection on Monday for a Wednesday video-conference. The network performs admission control tests and informs the client on the same Monday whether the desired resources will be available on Wednesday. Simulation results (described later in Chapter 4) show a synergy between resource sharing and advance reservation mechanisms, i.e. advance reservations help increase resource sharing acceptance gain, under the assumption that larger conferences will be more likely to make reservations in advance.

An important question involves trading off computational complexity with admission control efficiency: if two (or more) channels, advance booked for different (but possibly overlapping) time-intervals can share resource allocations, how does the network handle such sharing? There are at least three alternatives:

- Ignore such sharing relationships
- Make independent decisions (about whether to use individual or aggregate traffic specification) for each time interval
- Make the same decision for all time intervals

The first option would imply losing all resource sharing gains – this is clearly not desirable. If we make independent decisions for each time interval, we will obtain the best possible efficiencies in resource allocation; however, this would imply that at the interval boundaries, RCAP will have to contact RTIP to change between group and individual and aggregate traffic specifications. This introduces undesirably strict timing requirements. Also, this option would imply storing resource sharing information on a per-time-interval basis, and that would lead to a tremendous increase in the amount of state information to be maintained, as well as in the computational overhead associated with admission control. With the third option, we trade-off some potential resource allocation efficiencies to (a) avoid strict timing requirements, and (b) save on the state information maintained as well as the computational overhead for running admission tests. We chose this third option for our Suite 2 implementation.

## 2.7 Summary

*Resource sharing* exploits known relationships among related channels to allow network resources to be shared without sacrificing well-defined guarantees; most importantly, for large conferences with a bounded number of concurrent speakers, resource reservations do not increase with the number of potential speakers. Therefore, resource sharing is an important tool for providing real-time guarantees for large conferences.

We presented a scheme for sharing resource allocations between guaranteed performance connections in computer networks; this scheme provides a fully-distributed, low-overhead technique for implementing resource sharing. We evaluated the resource sharing performance gains by analysis and by simulations; these confirmed the intuition that resource sharing is very useful in saving network resources and quantified these gains. It achieves both higher connection acceptance rate and lower computational cost for admission control (than without resource sharing), while still providing guaranteed performance to the clients, *independent of the behavior of other, unrelated, traffic.* This performance evaluation also helped us in understanding how different factors (e.g., routing) impact these gains. As described in [61], we compared the analytical results with the simulation results; the comparison showed the simulation results correspond fairly well to the analytical values.

We then presented some of the interactions of resource sharing with other components of our multi-party real-time communication system; we described the interactions with the local admission control system, with the choice of traffic specification models, with the routing system, and with the advance reservations system. In Chapter 3, we will describe the interactions of resource sharing with resource partitioning mechanisms.

Resource sharing is a key component of our multi-party real-time communication system; in the next two chapters, we will describe two other important components: the resource partitioning system and the advance reservations system. As we will see in these chapters, our resource sharing mechanisms interact well with the resource partitioning and advance reservation mechanisms to improve overall system performance. We will then describe, in Chapter 5, how these components work together in our Tenet Real-time Protocol Suite 2; we will also present some measurements of our prototype implementation of Suite 2, which show that our techniques work very well in practice. We will also briefly describe the *channel groups* that we use to specify the resource sharing relationships, and contrast our approach with that taken recently by several other researchers, including the designers of RSVP[130, 82, 81, 36, 10] and ST2[114, 97, 106, 34, 35, 33, 66, 65].

# Chapter 3

# Resource partitioning

For real-time communication services to achieve widespread usage, it is important that network managers be allowed to control the services effectively. An important management capability concerns *resource partitioning*, i.e., distributing the different resources available at any given server (network node or link) among a number of partitions, where the admission control and establishment computations for a given connection need to consider only the connections in the same partition, and are completely independent of the connections accepted in other partitions. Resource partitioning is useful for a number of applications, including the creation of virtual private subnetworks and of mechanisms for advance reservation of real-time network services, fast establishment of real-time connections, and mobile computing with real-time communication.

In this chapter, we present resource partitioning: we describe admission control tests for resource-partitioned servers with four representative scheduling disciplines, Earliest-Deadline-First (EDD), First-In-First-Out (FIFO), Rate-Controlled-Static-Priority (RCSP), and Weighted-Fair-Queueing (WFQ), provide simulation results, and discuss the key implementation issues. Our simulations confirm the intuition that resource fragmentation losses due to resource partitioning are small, and that resource partitioning reduces the admission control computation overhead. An interesting result from the simulation experiments is that, under circumstances that arise naturally in multi-party communication scenarios, resource partitioning results in *higher* overall connection acceptance rates.

## 3.1 Introduction

We use the term *resource partitioning* to refer to the set of techniques and mechanisms that provide the network managers with the ability to distribute the different resources available at any network node or link among a number of partitions. In this sense, a "partition" is a virtual network in which real-time connections can be created; a partition consists of the resources allocated to it in various nodes and links of the physical network. In the sequel, the term "partition" will be sometimes used to refer also to the fraction of a node's or a link's resources allocated to the corresponding network-wide partition.

With resource partitioning, the admission control and establishment computations for a particular connection are independent of the connections accepted outside that connection's partition. This independence amounts to splitting the server into a number of sub-servers, each offering QoS guarantees only to the connections within it; the guarantees are valid as long as the admission tests and rate control schemes for all sub-servers are correct, and are independent of the resource reservation decisions and computations performed in other partitions. Figure 3.1 illustrates this issue: (a) shows a network node with three links; (b) shows the node model with one CPU server and one link server per outgoing link in each of the two partitions.

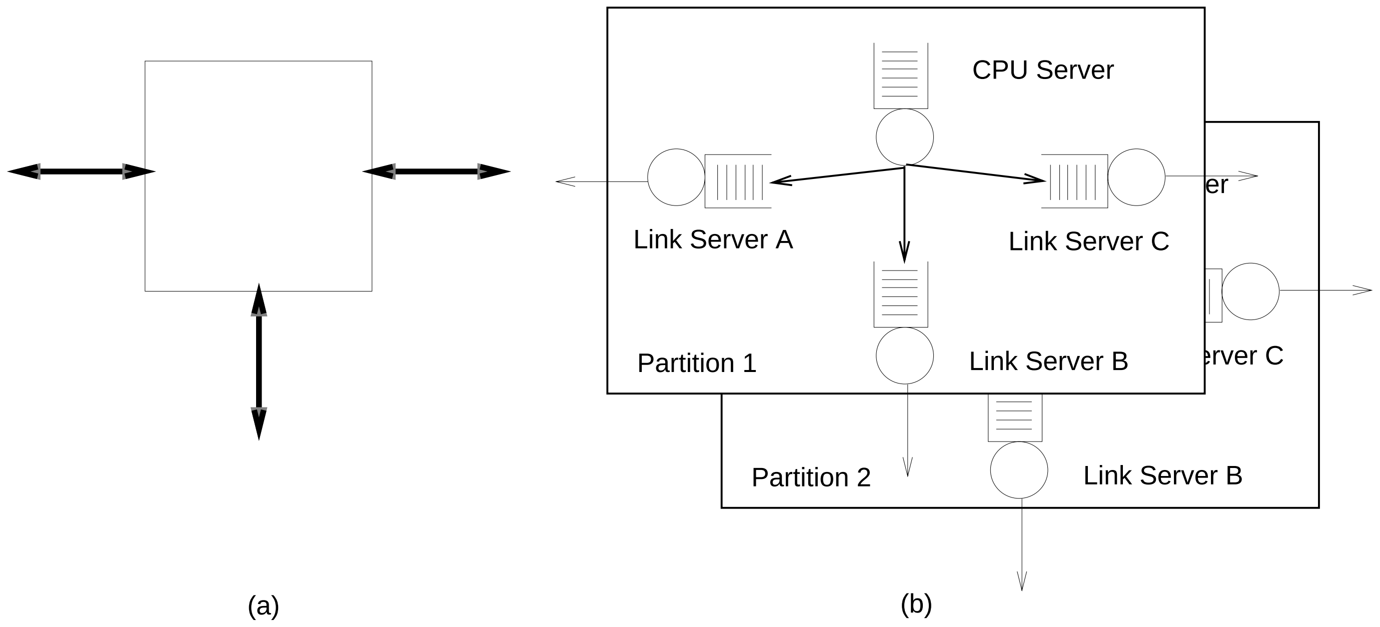

Figure 3.1: The server model for a node with three links and two partitions

This server/partition independence is very useful, as it can be used for the following:

- The different sub-servers can be used to form virtual private subnetworks.
- Network managers can keep a small fraction of resources for management and fault-handling traffic.
- A fraction of the network's resources may be kept for non-real-time traffic.
- The network can implement fairness constraints on network accessibility (preventing one set of clients from hogging up network resources).

In addition to advance reservations [47], which we will discuss in Chapter 4, other resource partitioning applications include fast establishment of real-time connections and support for mobile computing [57]. We will describe these applications in Chapter 7; the interested reader is referred to [45] for a more detailed description of these applications.

Many important concerns can be raised regarding resource partitioning: for instance, whether we can design efficient mechanisms for resource reservation, and whether these mechanisms can be designed for different scheduling disciplines and admission control procedures. The efficiency concern encompasses two related issues. First, the computational expense associated with admission control should not appreciably increase under resource partitioning (in fact, it is expected to decrease, due to reasons to be explained later). Secondly, any partitioning scheme is affected by *resource fragmentation losses*. For example, if a link can support 50 connections, and the resources at this link are divided into three equal partitions, then each partition can only support sixteen connections. Thus, the link can now only support 48 connections, and this corresponds to a fragmentation loss of two connections.

Note that, as will be seen in Section 3.2, the derivation of some admission tests for a partition is nontrivial, due to the same reason that makes it hard to derive those admission tests in general: while testing against the still available amounts of bandwidth and buffer space in a server is easy if the amounts of these resources required by the channel to be established are known, verifying the schedulability of packets in a bounded-delay context after the addition of a new channel can be more complicated.

In the following sections, we illustrate our approach to network resource partitioning in the framework of the Tenet real-time communication protocols [44]. As described before, these protocols are based on the *real-time channel* communication abstraction. The Tenet approach is connection-oriented and reservation-based: before a real-time channel can be used by its requester, it must be established (i.e., resources for the channel must be set aside along its route), so that the desired performance guarantees can be provided. While our design, simulations and implementations were done in the framework of the Tenet-style connection establishment, the principles are equally applicable to other connection setup protocols and techniques, including those followed by ST-2 [97], RSVP [130] and OPWA [108].

This chapter is organized in the following manner. In Section 3.2 we discuss resource-partitioning-oriented admission control tests for four packet scheduling disciplines: Earliest Deadline First (EDD), First In First Out (FIFO), Rate Controlled Static Priority (RCSP), and Weighted Fair Queueing (WFQ). Section 3.3 presents a simulation-based evaluation of resource partitioning algorithms in a multi-party communication environment. We discuss implementation issues in Section 3.4, and Section 3.5 concludes this chapter.

## 3.2 Resource partitioning tests

What parts of an admission control algorithm do we need to modify to make network resources partitionable according to our approach? Our ideal objective is to subdivide a network (or, more generally, an internetwork) into a certain number of virtual networks (or internetworks), each one of which can be treated totally independently of the others, even though they all share the same hosts, nodes, and links. This is equivalent to saying that we would like to confine our admission tests to the resources assigned to, and the channel created within, the partition to which a new channel is requesting admission, without in any way involving the channels and the resources belonging to the other partitions. Network resources can be partitioned in this sense if, given a number of partitions and suitable admission tests to each partition, we can prove that all channels established in all partitions always satisfy the admission tests for the full network. In this section, we show that this goal can be reached for many packet-scheduling disciplines; we present the *partitioned* admission tests for four packet scheduling disciplines: EDD, FIFO, RCSP, and WFQ. By examining these disciplines, we show that our resource partitioning techniques are general, as they apply to a wide spectrum of scheduling policies, and also to internetworks with heterogeneous (e.g., multi-vendor) nodes, as well as to nodes modeled with several servers[1] using different scheduling disciplines. EDD is an excellent but relatively expensive policy; RCSP and WFQ are good and less expensive; RCSP is a static-priority discipline, WFQ is a member of the round-robin family; FIFO is not very good but probably the cheapest to implement. Since some of the admission tests depend on the packet scheduling discipline, the corresponding partitioned tests are also scheduler-dependent.

Note that, to be accepted, a request for a new channel must pass also, besides the tests we discuss below, a buffer space test in each server [45]. This test is very easy to derive, and will be omitted for the sake of brevity throughout our discussion.

### 3.2.1 Resource partitioning in an EDD-scheduled server

In an EDD-scheduled server, an established channel $k$ with a deterministic delay bound is characterized by $t_k$, the maximum service time for any packet belonging to this channel, and by $d_k$, the local delay bound (which is the maximum amount of time that any packet on this channel will stay in this server) [117]. In addition, the load at the server is characterized by $t^*$, the maximum service time for any packet (real-time or best-effort) serviced. We assume, for brevity of explanation, that the local admission control process maintains the list of already established channels sorted by non-decreasing local delay bounds ($d_i \geq d_j$ if $i \geq j$). To a new resource request $R$ with maximum packet service time $t_{new}$ [2] we can assign a delay bound $d_{new}$, inserting the new channel into the list without any violations of the local delay bounds if, after adding this new connection,

$$d_i \geq \sum_{l=1}^{i} t_l + t^*, \tag{3.1}$$

where the index $i$ goes over all real-time channels in the server, including the new one. [49].

The test ensures that, when a packet arrives over a channel $k$, the maximum amount of time it could possibly wait (before being transmitted) is bounded above by the delay bound $d_k$.

To determine the form of the delay bound computation if we have resource partitioning, we need to introduce a few definitions and theorems.

[1]As mentioned before, a server is a network component that has resources to be allocated, e.g., a link.

[2]$t_{new}$ depends on the maximum packet size and the service rate at the server.

**Definition 3.1** *A schedule $S$ in an EDD server is a set of channels $C_1, C_2, ..C_m$ with local delay bounds $d_1, d_2, ..d_m$ such that $d_1 \le d_2 \le .. \le d_m$.*

**Definition 3.2** *Schedule $S$ is "acceptable for $\delta$" in an EDD server if*

$$d_k \ge \frac{1}{\delta} \sum_{l=1}^{k} t_l + t^* (k = 1, 2, ..m), \qquad (3.2)$$

*where $0 \le \delta_j \le 1$, $t_k$ is the maximum service time for a packet on channel $C_k$ in the server, and $t^*$ is the maximum service time that any packet can have in this server.*

**Definition 3.3** *Two or more schedules are said to be disjoint if all their pairwise intersections are empty.*

**Lemma 3.1** *If schedule $S$ is acceptable for $\delta$ in an EDD server, then $S$ is acceptable for all $\lambda$ such that $\delta \le \lambda \le 1$.*

**Proof**: Follows directly from (3.2).

**Theorem 3.1** *If disjoint schedules $S_1, S_2, ..S_r$ are acceptable for $\delta_1, \delta_2, ..\delta_r$, respectively, in an EDD server, so that*

$$\sum_{h=1}^{r} \delta_h \le 1, \qquad (3.3)$$

*then the combined schedule $S = S_1 \cup S_2 \cup .. \cup S_r$ is acceptable for $\sum_{h=1}^{r} \delta_h$.*

**Proof**: We shall prove that the theorem holds for two disjoint schedules; the extension of the proof to $r$ disjoint schedules is trivial. Let $C_k$ be the $k$-th channel in the combined schedule $S$. Without loss of generality, we assume that the channel $C_k$ was the $k1$-th channel in schedule $S_1$. Then, $k2 = k - k1$ is the number of channels in schedule $S_2$ that precede $C_k$ in $S$. Since $S_1$ is acceptable for $\delta_1$, we must have

$$d_{k1} \ge \frac{1}{\delta_1} \sum_{l=1}^{k1} t_{k1} + t^*; \qquad (3.4)$$

since $S_2$ is acceptable for $\delta_2$, and $C_k$ follows the channel with delay bound $d_{k2}$ in $S$,

$$d_{k1} \ge d_{k2} \ge \frac{1}{\delta_1} \sum_{l=1}^{k2} t_l + t^*; \qquad (3.5)$$

Thus,

$$d_{k1} \ge \frac{1}{\delta_1 + \delta_2} \left( \sum_{l=1}^{k1} t_l + \sum_{l=1}^{k2} t_l \right) + t^*. \qquad (3.6)$$

**Q.E.D.**

We now state the admission control tests for an EDD server with resource partitioning. Consider an EDD server with partitions $P_1, ...P_n$, partition $P_s$ being allocated a fraction $\delta_s$ of the "schedulability" (or "delay") resources, so that $\sum_{s=1}^{n} \delta_s \leq 1$. The EDD server will guarantee this delay bound to all packets of partition $P_s$ (with allocation $\delta_s$) if

$$d_h \geq \frac{1}{\delta_s} \sum_{l=1}^{h} t_l + t^*, \tag{3.7}$$

where the index $h$ goes over all channels in partition $P_s$.

Note that, for EDD servers, there is also a separate bandwidth test, whose adaptation to the partitioning case is, however, trivial and will not be described here. The interested reader can find the description and the proof in [45].

### 3.2.2 Resource partitioning in a FIFO server

We now describe the admission control test for a FIFO server without resource partitioning. In a FIFO server, all real-time connections are assigned the same local delay bound, say $d$. Let traffic over a real-time connection be characterized at the network layer by the quadruple $(Xmin, Xave, I, Smax)$, where $Xmin$ is the minimum interpacket interval, $Xave$ is the minimum average interpacket interval, $I$ is the averaging interval, and $Smax$ is the maximum packet size. To a new resource request $R$ with traffic specification $(Xmin, Xave, I, Smax)$ we can assign local delay bound $d$ without violating the delay bounds of existing connections if, after adding this new connection,

$$\sum_i \lceil \frac{d}{Xmin_i} \rceil * Smax_i + Smax^* \leq d * ServiceRate. \tag{3.8}$$

where the index $i$ goes over all real-time channels at that server, $Smax^*$ is the size of the largest packet (either real-time or best-effort) that is to be serviced by this server, and $ServiceRate$ is the server speed (say in bps).

The test [125] ensures that, when a packet arrives, the maximum amount of time it could possibly wait (before being transmitted) is bounded above by the delay bound $d$ associated with that server. It is easy to see that

$$0 < Smax^* \leq d * ServiceRate. \tag{3.9}$$

We now introduce the FIFO admission control tests with resource partitioning.

**Theorem 3.2** *Consider a FIFO server with delay bound $d$, and partitions $P_1, ...P_n$, with partition $P_s$ allocated a fraction $\delta_s$ of the server resources such that*

$$\sum_{s=1}^{n} \delta_s \leq 1. \tag{3.10}$$

*The FIFO server will guarantee delay bound d to all packets of partition $P_s$ (with allocation $\delta_s$) if*

$$\sum_i \lceil \frac{d}{Xmin_i} \rceil * Smax_i + Smax^* * \delta_s \leq d * ServiceRate * \delta_s, \tag{3.11}$$

*where the index i goes over all channels in partition $P_s$.*

**Proof** We say that the test in (3.11) is valid if it only admits channels to the given partition that always satisfy the test in (3.8) applied to the entire population of channels. The validity of (3.11) can be observed by adding the admission control tests over all partitions to obtain

$$\sum_i \lceil \frac{d}{Xmin_i} \rceil * Smax_i + Smax^* * \sum_{s=1}^{n} \delta_s \leq d * ServiceRate * \sum_{s=1}^{n} \delta_s, \tag{3.12}$$

where the index $i$ goes over all connections in the FIFO server. Substituting (3.9) and (3.10) in (3.12), we obtain

$$\sum_i \lceil \frac{d}{Xmin_i} \rceil * Smax_i + Smax^* \leq d * ServiceRate, \tag{3.13}$$

that is, the resource-partitioning test in (3.8). **Q.E.D.**

### 3.2.3 Resource partitioning in an RCSP server

Hui Zhang and Domenico Ferrari designed the RCSP packet scheduling discipline and described the admission control tests for RCSP servers (without resource partitioning) in [125]. We first provide a brief introduction to RCSP, and then we describe admission tests for *partitioned* RCSP servers.

#### A brief introduction to RCSP

Figure 3.2 shows an RCSP server for a node with $x$ input links and a single output link; additional links can be added by replicating the scheduler portion of the server. Only the handling of real-time traffic is shown in the figure; non-real-time traffic is collected from the input links into per-output-link queues, each of which has the lowest static priority among the queues for the corresponding outgoing link.

An RCSP server has two components: a rate controller and a static-priority scheduler. The rate controller shapes the input traffic from each connection (so that the packets do not violate the traffic specification when they go into the scheduler); the scheduler orders the transmissions of the packets from all connections. By neatly separating the rate-control and delay-control functions in this manner, RCSP achieves flexibility in allocation of delay and bandwidth, as well as simplicity of implementation.

Conceptually, a rate controller consists of a set of regulators corresponding to each of the connections traversing the switch; each regulator is responsible for shaping the input

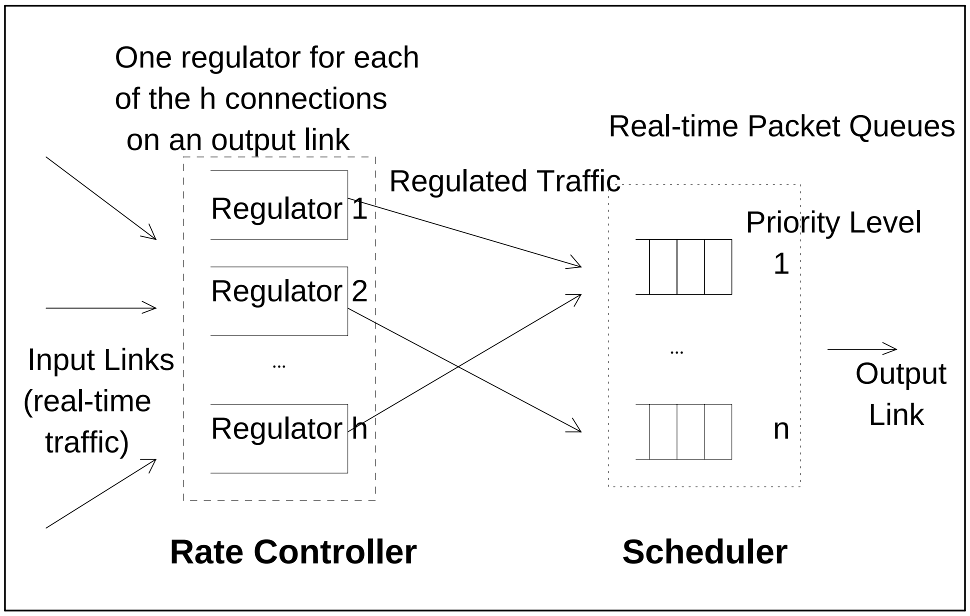

Figure 3.2: An RCSP server (courtesy Hui Zhang)

traffic of the corresponding connection into the desired traffic pattern. Regulators control the interactions between switches and reduce or eliminate jitter. Regulators achieve this control by holding data packets for the appropriate amount of time before handing them to the scheduler.

The scheduler services packets using a non-preemptive static-priority discipline:

- when the server chooses the next packet to transmit, the packet at the head of the highest-priority non-empty real-time queue is chosen; the packets in each real-time queue are serviced on a first-come-first-served basis;
- non-real-time packets are transmitted only when there are no real-time packets in the scheduler;
- the transmission of a lower-priority packet is not preempted by the arrival of a higher-priority packet.

### RCSP admission control tests

We first describe the admission control tests for RCSP servers without resource partitioning. Consider an RCSP scheduler with $L$ priority levels, and with $d_i$ as the local delay bound associated with priority level $i$. To a new resource request $R$ with traffic specification $(Xmin, Xave, I, Smax)$ we can assign a local delay bound $d_m$ (the delay bound associated with priority level $m$) without violating the delay bounds of existing connections if, after adding this new connection, for all priority levels $l$, $1 \leq l \leq L$,

$$\sum_i \lceil \frac{d_l}{Xmin_i} \rceil * Smax_i + Smax^* \leq d_l * ServiceRate. \tag{3.14}$$

where the index $i$ goes over all channels at or above the priority level $l$, and $Smax^*$ is the largest packet size that is to be serviced by this server[3].

Intuitively, the tests ensure that, when a packet arrives (for a connection at priority level $l$), the maximum amount of time it could possibly wait (before being transmitted) is bounded above by the delay bound $d_l$ associated with that priority level.

We now introduce the admission control tests for RCSP servers with resource partitioning. Consider an RCSP scheduler with $L$ priority levels, a partition with a fraction $\delta$ of the server's resources, and with $d_i$ as the delay bound associated with priority level $i$. We assume, for simplicity, that the same fraction $\delta$ that characterizes a partition in a server applies to all $L$ priority levels (each partition may have channels assigned to all priority levels). The RCSP server will guarantee these delay bounds to all packets of this partition if, for all priority levels $l$, $1 \leq l \leq L$,

$$\sum_i \lceil \frac{d_l}{Xmin_i} \rceil * Smax_i + Smax^* * \delta \leq d_l * ServiceRate * \delta, \tag{3.15}$$

where the index $i$ goes over all channels in that partition, as long as the sum of the fractions $\delta$ of resources allocated to all partitions does not exceed unity.

**Theorem 3.3** *Consider an RCSP server with delay bounds $d_1, d_2, ..., d_L$, and partitions $P_1, ...P_n$, with allocation $\delta_s$ to partition $P_s$. Under the condition:*

$$\sum_{s=1}^{n} \delta_s \leq 1, \tag{3.16}$$

*the RCSP server will guarantee these delay bounds to all packets of each partition $P_s (s = 1, 2, ..., n)$ if the tests in (3.15) are satisfied.*

**Proof:** From the resource partitioning tests (3.15) above, we know that, for all levels $t, 1 \leq t \leq l$, for all partitions $P_s, 1 \leq s \leq n$, the following condition holds:

$$\sum_i \lceil \frac{d_l}{Xmin_i} \rceil * Smax_i + Smax^* * \delta_s \leq d_l * ServiceRate * \delta_s, \tag{3.17}$$

where the index $i$ goes over all channels in partition $P_s$ at or above the priority level $t$, and $Smax^*$ is the largest packet size that is to be serviced by this server.

At level $t$, we add up tests (3.15) for all partitions, and we obtain

$$\sum_i \lceil \frac{d_l}{Xmin_i} \rceil * Smax_i + Smax^* * \sum_{s=1}^{n} \delta_s \leq d_l * ServiceRate * \sum_{s=1}^{n} \delta_s \tag{3.18}$$

[3]Note that this inequality is the same as (3.8), but $d$ is replaced by $d_l$, and index $i$ goes over all channels at the same or higher priority.

where the index $i$ now goes over *all* channels in *all* partitions at that server at or above the priority level $t$, and $Smax^*$ is again the largest packet size that is to be serviced by this server.

Since we have

$$Smax^* \leq d_l * ServiceRate, \tag{3.19}$$

for all levels $l, 1 \leq l \leq L$, substituting (3.19) and (3.16) into (3.18), we obtain

$$\sum_i \lceil \frac{d_l}{Xmin_i} \rceil * Smax_i + Smax^* \leq d_l * ServiceRate, \tag{3.20}$$

that is, the resource partitioning tests in (3.14). **Q.E.D.**

### 3.2.4 Resource partitioning in a WFQ server

We briefly describe the admission control test for a simple Weighted-Fair-Queueing (WFQ) server with and without resource partitioning. In WFQ, the server assigns, to each real-time channel $i$ being requested, a weight $\phi_i$ such that

$$\sum_j \phi_j \leq 1, \tag{3.21}$$

where $j$ goes over all channels in that server.

Depending on the assigned weight $\phi_j$ and the channel traffic parameters, the server computes the performance parameters it can offer to this new channel [88].

Resource partitioning imposes a small change to this procedure. Instead of the inequality (3.21), for a partition $P_s$ with a fraction $\delta_s$ of resources, we have

$$\sum_j \phi_j \leq \delta_s \tag{3.22}$$

where $j$ goes over all channels in that partition, including the one being requested.

## 3.3 Simulations

In the previous section, we demonstrated that our resource partitioning techniques can be applied to many scheduling disciplines. In this section, we show that these techniques are useful and efficient.

We performed many simulation experiments to evaluate the performance of the resource partitioning algorithms. The simulation experiments confirmed our intuitive feelings and expectations about the system's behavior under resource partitioning. Our goal was to make the experiments as real-life as possible, so that we could confidently predict the behavior of our implementation of resource partitioning in the Tenet Protocol Suite 2 [60]. For example, we used the NSFNET backbone network topology (shown in Figure 2.8) in our simulations. We assumed the rate of each link to be 45 Mbps, the propagation delay along the diameter to be 40 ms, and we also assumed that we could allocate up to 80% of the resources to real-time communication, so that non-real-time traffic would get at least 20% of the total resources. We made the amount of buffer space in each server large enough that the bandwidth or processing power was the limiting resource in all servers and all scenarios.

In all experiments, the sources and destinations for the channels were chosen uniformly and independently among the network nodes. To keep comparisons meaningful, we only considered a single type of traffic stream ( i.e., a compressed video stream with a peak rate of 1 Mbps, 30 frames per second, and four data packets per frame), and destinations with identical performance requirements ( i.e., end-to-end delay bound 400 ms); the average data rate did not matter, because the admission tests in our simulations used peak-rate bandwidth allocation. The main metric we adopted for evaluation and comparison was the *acceptance ratio*, defined as the ratio between the number of destinations reached with resource partitioning and the number of destinations reached without resource partitioning. We were also interested in comparing the computational overhead associated with admission control, with and without resource partitioning. For this, we adopted the *overhead ratio* metric, which, as in Chapter 2, is defined as the ratio between the computational overhead with resource partitioning and without resource partitioning.

In this section, we present two sets of simulation experiments: one characterized by homogeneous requests, and the other one by requests of two types, for unicast channels and for conferences using multicast channels.

### Homogeneous requests

In the first set of experiments presented here, we ran our simulations with simplex unicast connections alone; with these connections, quantitative comparisons using the metrics described above are particularly easy to make.

We compared the following two scenarios:

- 300 simplex unicast connections, all in the same partition, which is allocated 80% of the network's bandwidth; we call this the *"without resource partitioning"* case; and

- two partitions with 150 simplex unicast connections each, and varying partition allocations, so that the total resource allocation for these partitions equals 80% of the network's bandwidth; this is the *"with resource partitioning"* case.

We deliberately chose this workload to saturate the network, because we wanted to observe the network's behavior under heavy real-time load.

In all figures, we report on the horizontal axis the fraction $f$ of the total resources that is allocated to one of the partitions. The other partition's allocation is $100(0.8 - f)\%$ of the total resources.

As we mentioned in Section 3.1, with resource partitioning we can expect fragmentation losses; in Figure 3.3, we observe fragmentation losses of up to about 20%, depending on the relative resource allocations to the partitions. This graph also verifies that the

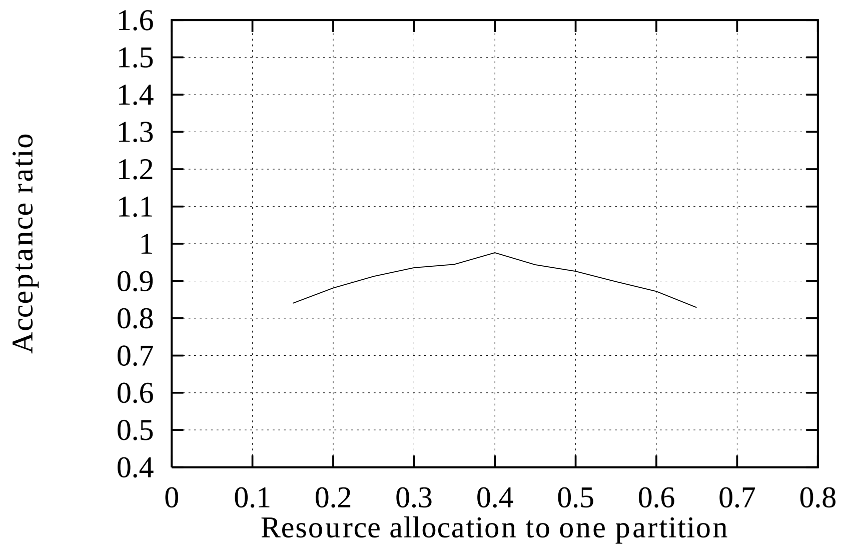

Figure 3.3: Acceptance ratio vs. relative partition allocation for unicast channels

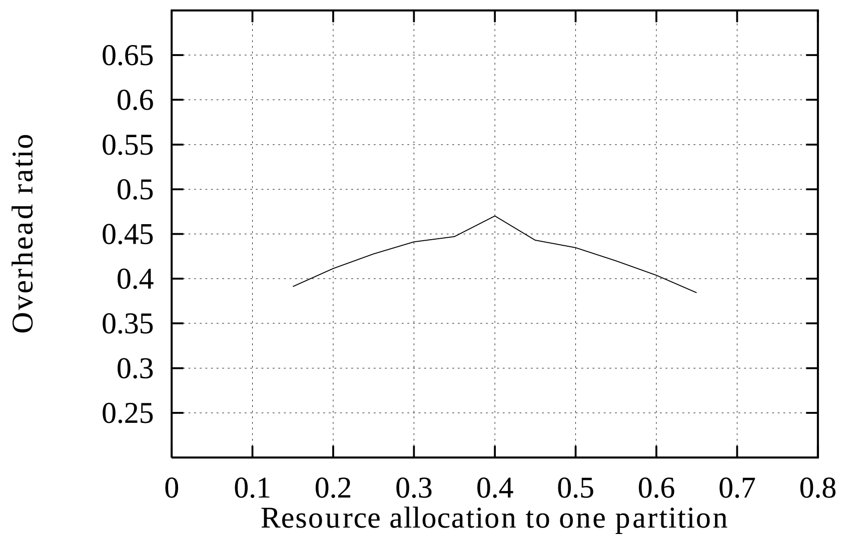

Figure 3.4: Computational overhead vs. relative partition allocation for unicast channels

partitioning scheme may be made to work fairly well, since, if the allocations are appropriately chosen (i.e., if they are suitable for the actual requests), the fragmentation losses are minimal (about 2-3%). As all channels are identical and equally distributed between the two partitions, the best choice is that of identical allocations. Note that this curve is, as expected, symmetrical with respect to the vertical line at $f = 0.4$.

Resource partitioning reduces the computational overhead associated with admission control because, during admission control tests for a new connection, we only have to consider other connections within the same partition; without resource partitioning, we would have to consider all the connections at that server. In this experiment, we expected resource partitioning to lead to a reduction of about 50% in computational overhead, and the simulation results in Figure 3.4 verified this intuition.

### Heterogeneous requests: multi-party communication

In the second set of experiments, we considered heterogeneous requests. Here, some requests were for simplex unicast connections; the others were for conferences, where the participants could *share* resources [62]. We considered the case where the conference requests were all served by one partition, while the unicast connection requests were served by the other partition. This segregation may be a natural consequence of some aspects of multi-party communication, for instance advance reservation requirements [47].

We compared the following two scenarios:

- 150 simplex unicast connections and 50 10-person conferences, all in the same partition, which was allocated 80% of the network's resources; as above, this is called the *without resource partitioning* case; and

- two partitions, with 150 simplex unicast connections in the first partition, and 50 10-person conferences in the second partition; partition allocations were varying, but their total resource allocation was always equal to 80% of the network's resources.

The graph in Figure 3.5 shows an interesting phenomenon, which could be interpreted as the reverse of fragmentation. There is a fairly large region in which the overall channel acceptance ratio is higher than one, i.e., the acceptance rate is higher with resource partitioning than without resource partitioning. In these simulations, we observed reductions in computational overhead similar to those obtained in the first set of experiments [58].

The above-observed phenomenon (increased channel acceptance) is easily explained in the following manner. First, resource allocation requests have varying efficiencies in using resources. As we saw in Chapter 2, with resource sharing, the resource requirements do not increase with allocation requests for additional channels [62]; this implies that conference requests are more efficient in using resources than isolated connections. Second, as we mentioned before, partitioning provides protection for allocation among partitions; in this case, partitioning ensures that the resources allocated to the first partition will only be used for conferences, and not for isolated connections. As the conferences use resources more efficiently, the acceptance gains with conferences may be large enough to offset and

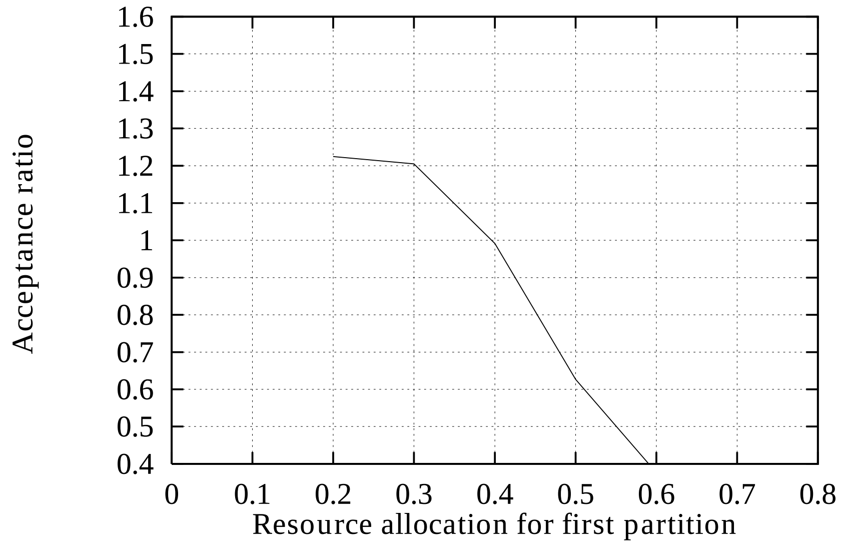

Figure 3.5: Acceptance ratio vs. relative partition allocation for the multi-party communication scenario. The first partition is for unicast connections.

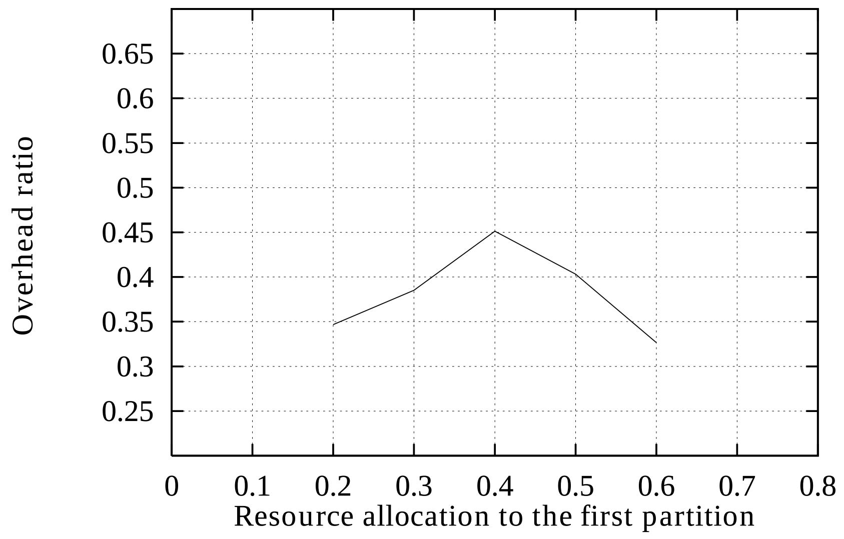

Figure 3.6: Computational overhead vs. relative partition allocation for the multi-party communication scenario.

more than compensate for the fragmentation losses that we observed in the previous set of experiments.

Resource partitioning leads to higher connection acceptance when the network is overloaded (i.e., when the total request for resources exceeds the supply, and thus, the admission control has to refuse some requests) and when the partitions that accommodate resource-sharing requests do not service a large number of isolated connections.

## 3.4 Discussion

In the previous sections, we provided the admission control tests, with resource partitioning, for various packet scheduling disciplines, including EDD, FIFO, RCSP, and WFQ. We also presented the results of our simulation experiments; these confirm the intuition that resource fragmentation losses due to resource partitioning are small, that resource partitioning reduces admission control computational overhead, and that under circumstances that arise naturally in multi-party communication scenarios, resource partitioning leads to *higher* overall acceptance rates.

These simulations also showed that, by choosing wrong resource allocations for different partitions, we can seriously degrade the system performance; inappropriate resource allocations can lead to significantly *lower* acceptance rates. This problem can be handled by making dynamic changes; these can be

- moving channels between partitions,
- "borrowing" resources between partitions,
- dynamically changing partition resource allocations.

Resource partitioning is an integral component of a multi-party real-time communication system; for good overall system performance, it is important that the resource partitioning mechanisms interact well with the other components of the system. including the routing system and the mechanisms for resource sharing.

In this section, we describe the interactions of resource partitioning with the resource sharing mechanisms and the routing system; we also describe dynamic adaptation mechanisms that ensure high connection acceptance rates.

### Resource sharing

The simulations with multi-party communication described in Section 3.3 showed that the resource partitioning mechanisms interact very well with the resource sharing mechanisms; indeed, the increase in acceptance rate, which we observed in these simulations, arose directly from the manner in which these mechanisms worked together. The protection provided by the resource partitioning mechanisms helps protect the *conference channels*, which, due to resource sharing, are more efficient in using network resources.

There is at least one other interesting scenario. In some applications (e.g., distributed simulations [10]), channels belonging to different sessions can *share* resource allocations. What do we do when we learn that two (or more) channels can share resources, though they belong to separate partitions? We can do one of the following:

- ignore resource sharing relationships across partition boundaries;
- if resource sharing can be used to save resource allocations, *move* all such channels to one partition;
- allow different partitions to contribute their respective *share* to the aggregate group allocation.

The first approach is the simplest to implement, though it may lead to some inefficiencies in resource allocation. The other approaches lead to more book-keeping and add complexity to the code, especially when channels are torn down.

In our initial implementation, we have taken the first approach mainly because it is the simplest to implement. In the future, it would be interesting to investigate the other approaches as well.

### Routing

The routing subsystem is a key component of any multi-party real-time communication system. Good, efficient, and robust distributed routing techniques are hard to design; the complexity increases due to real-time performance bounds and also due to multicasting. In addition, as we discussed in Chapter 2, resource sharing has a significant impact on the routing mechanisms. Resource partitioning adds to this complexity.

With resource partitioning, we create multiple independent subnetworks; this affects routing in two manners:

- The routing service can treat different partitions (*subnetworks*) independently. This implies that the routing system must keep information on a per-partition basis. This implies more bookkeeping. In addition, the routing system must re-compute state information if the network moves channels between partitions.
- Changing partition allocation amounts to changing the characteristics (capacity) of the corresponding servers, and the routing service must re-compute state information accordingly.

### Changing the partitions of existing channels

In our system, admission control can be viewed as an accounting system that ensures that the network has enough resources to support the various requests for guaranteed-performance connections. In this view, resource partitioning can be seen as dividing a server's resources among the partitions for accounting purposes, and channel requests as identifying the account that should be debited. In this view, it is easy to see that by appropriate accounting, we can transfer resource requests between the servers (just charge the resources to another account; if this is feasible, just release the resources from the previous account). Since resource partitioning only affects connection establishment (and is completely invisible to the packet scheduling and data transfer functions), existing channels can change partitions without affecting data transmission and delivery.

However, changing partitions raises two related issues: access control and the interface for specifying these changes. Clearly, for changing over to a new partition, the clients must have authorization to use the resources of that partition. Also, the network has to define interfaces for supporting this change. In particular, the establishment and partitioning techniques can permit a channel to traverse a network, while, depending on resource availability, charging resources to different partitions on different nodes along the channel route.

### "Borrowing" resources between partitions

As the simulation experiments in Section 3.3 have shown, connection acceptance can be reduced by the wrong choice of the amounts of resources allocated to each partition; the acceptance ratio goes down whenever one partition refuses channel requests because it is already "overloaded" while there exists another partition in which the resource supply exceeds the demand. One solution would be to support fast dynamic changes in partition allocation; we will discuss it later in this section. Another possible solution would be to permit "resource borrowing" between partitions (as in CBQ [121]); in this case, the overloaded partition would borrow resources from the underloaded one, with the promise that the resources would be returned if the second partition needed them.

Such borrowing would be feasible if we could predict that the second partition will not need the resources for some time in the near future, and that the resources are lent only for that time interval (of course, the overloaded partition can attempt borrowing resources again at the end of this time interval). Unfortunately, this solution does not work well with service guarantees; when the network accepts a channel request, it does not know when the channel resources will be released (except for advance reservation requests, which we will describe in Chapter 4). Also, the network cannot guarantee that no channel requests will arrive for a partition in the near future. Due to these reasons, such borrowing appears infeasible for guaranteed services.

There is one case in which the network can guarantee these "predictions". For advance-reserved connections, the network service provider(s) can set up, as a policy decision, the minimum advance notice period; in this case, the network can confidently claim that no additional advance reservation requests will arrive for using resources for this minimum advance notice period, and it can therefore permit such "borrowing" from a partition of advance channels.

### Dynamically changing partition resource allocations

As the simulations in Section 3.3 show, choosing incorrect resource allocations can lead to underutilization of resources. One way to eliminate these losses would be to dynamically permit changes in partition resource allocations, so that setup errors could be corrected, and that the network could adapt to dynamic changes in resource allocation requests.

Partitions may be owned by the network or by a client organization. A real-time service cannot allow uncontrolled changes in partition resource allocations; the owner of a partition has to decide if the partition's resource allocations can be reduced, and, if so, what are the limits to the reduction. The contract to be stipulated between a partition owner and the network managers may include clauses that will influence the destinations of the resources possibly released by the partition's owner. For example, the owner of partition A might decide that its allocation can be reduced by no more than 10% if partition B needs extra resources, and no more than 5% if partition C needs extra resources; also, that no reduction is allowed for any other partitions. The network must provide an interface that will allow network managers to specify these constraints. Also, a request for dynamic reduction of the resource allocation for a partition may not succeed if the needs of the existing channels in that partition would exceed the reduced resource allocations.

We will discuss these mechanisms in Chapter 5.

## 3.5 Conclusions

For real-time communication services to achieve widespread usage, it is important that network managers be allowed to control the services effectively. Resource partitioning provides one such important capability.

In this chapter, we described our techniques for resource partitioning in real-time networks. In our mechanisms, the partitioning computations are limited to channel establishment time; per-packet scheduling and data forwarding are not affected by partitioning. These resource partitioning techniques apply to many scheduling disciplines; we presented partition-oriented admission control algorithms for EDD, FIFO, RCSP and WFQ packet schedulers. We also presented the results of our simulation experiments; these verified the usefulness of our techniques. These simulations also showed that resource partitioning can substantially reduce the computational overhead associated with admission control for real-time connections. Also, under circumstances like those described in Section 3.3, resource partitioning techniques result in higher overall connection acceptance ratios.

Resource partitioning is an integral component of our multi-party real-time communication system; it is useful for many applications, including the creation of virtual private subnetworks and of mechanisms for advance reservation of real-time network services, fast establishment of real-time connections, and mobile computing with real-time communication. In the next chapter, we will describe how resource partitioning mechanisms work with the advance reservation mechanisms. In Chapter 7, we will describe some of the other applications.

# Chapter 4

# Advance reservations

In the previous two chapters, we discussed *resource sharing* and *resource partitioning*; resource sharing mechanisms provide efficient support for large-scale conferences while resource partitioning provides an effective tool for the network managers to control and distribute resource allocation among the various network service clients. In this chapter, we will discuss the third cornerstone of out multi-party real-time communication research: advance reservations. The ability to reserve real-time connections in advance is essential in all distributed multi-party applications (i.e., applications involving multiple human beings) using a network that controls admissions to provide good quality of service.

Providing advance reservations raises many important issues that encompass all aspects of multi-party real-time communication. For example, advance booking requires modifications in the client-service interface, the admission tests system, and the routing system, among others. We will discuss each in turn in this chapter. We will first discuss the requirements of the clients of an advance reservation service, and then describe a distributed design for such a service. The description will be in the framework of the Tenet Suite 2, which offers advance reservation capabilities to its clients based on the principles and the mechanisms described here. Simulation results providing useful data about the performance and some of the properties of these mechanisms are also presented. This chapter describes a viable approach to constructing an advance reservation service within the context of the Tenet Suites as well as that of other solutions to the multi-party real-time communication problem.

We have organized this chapter in the following manner. Section 4.1 motivates the current research in advance reservations, while Section 4.2 discusses the service requirements for advance reservations. In Section 4.3, we describe the distributed advance reservations mechanisms we have designed for, and are implementing in, the Tenet Suite 2 [60]. The principles on which our mechanisms are based, however, are easily portable to other approaches and protocols for real-time communication. Section 4.4 describes several important system issues, including the interaction of our advance reservation mechanisms with those for resource partitioning, as well as with the routing system. We also present several simulation results in Section 4.5, and conclude this chapter with a brief summary in Section 4.6.

## 4.1 Motivation

Some of the important multimedia applications of integrated services networks require that advance reservations be possible. The clients who wish to set up multimedia multi-party meetings (i.e., meetings involving multiple human beings) need to schedule those meetings in advance to make sure that all or most of the participants will be able to attend; at the time the meeting is scheduled, they must also be certain that the network connections and the other resources required will be available when needed and for the entire duration of the meeting. Unfortunately, distributed multimedia applications must be supported by real-time communication services, which are to provide the necessary quality-of-service (QoS) guarantees, and these services cannot admit an arbitrary number of connections. Thus, there is no guarantee that the resources for a pre-scheduled meeting will be available at the time the meeting is expected to start, unless they can be reserved in advance.

To our knowledge, advance reservation services are not available within any of the existing schemes for real-time communication (see for example [2, 9, 18, 97, 102, 130]). For example, in the client-service interface of the Tenet Suite 1 [2], there is no way a client can request the establishment of a real-time channel in advance. At any time before the beginning of a conference, a request could arrive that is accepted and that saturates the real-time capacity of one or more of the network's resources; this allocation, which cannot be prevented in any practical and efficient way[1], may preclude the establishment of one or more of the channels on which the conference depends, thereby causing the attempt to set up the conference to fail. Nor is it possible to predict when the resources needed by the conference will all be available, as real-time channels are to be established as soon as possible and for an indefinite duration.

We address here the problem of extending the Tenet scheme to allow for advance reservations of real-time channels. Our study has been performed within the context of a profound revision of the Tenet scheme, which has resulted in the design and development of a second-generation protocol suite, the Tenet Suite 2. Since this suite has been built to provide effective support to multi-party applications, the advance reservation service must be regarded as one of its essential new features.

## 4.2 Client requirements

The only true requirement network clients with multi-party applications have, in the area we are investigating here, is that they be allowed to specify in advance their needs in terms of real-time channels as though these channels were to be created immediately, and to obtain in this way a guarantee that the resources for those channels will be available at the future time they have specified. Clients will accept the necessity to reserve channels in advance if they can convince themselves that this is the only way to avoid the risk of partial (or total) rejection of their requests at the time they need to use the network.

[1]An alternative approach would be to provide pre-emption of existing channels; however, we believe that a good service should not be pre-emptible.

The service model in the existing proposals and realizations of real-time communication services, including that in the Tenet Suite 1 [2], assumes that real-time channels are requested (and established) for an indefinite duration. Clients are not asked to specify for how long such channels (to be called *immediate channels* in the sequel) will be alive, and this non-negligibly simplifies their tasks. When advance reservations are introduced into such a service, the provider has to do some planning for future allocations of resources, and this planning would be easier if the expected durations of the channels were known. A limitation of this duration would also allow more clients to reserve channels in advance, thereby increasing the sharing and the utilization of the resources. This modification of the service model for channels reserved in advance (henceforth to be called *advance channels*) is consistent with the practice of booking other types of facilities, for example, meeting rooms, which may never be reserved for an indefinite amount of time. For this reason, clients should be expected to accept this service model and conform to it without too much difficulty, especially if negotiating an extension of the channel's duration is sufficiently easy and inexpensive.

The same meeting-room analogy can be used to argue that, if the service provider found it useful to adopt a coarse granularity for time, i.e., to accept only starting times and durations that are integral multiples of, say, five minutes, clients would find it fairly easy to conform. Similarly, clients would probably accept, though perhaps not enthusiastically, reasonable values for the minimum and maximum advance notice with which reservation requests can be submitted (e.g., not less than one hour and not more than six months) if such limits were imposed by the provider.

Even with advance reservations, there is the possibility that a request be rejected. The significant difference with respect to the case in which a request for the immediate creation of a channel is rejected is that there is still time to reschedule or cancel the meeting without any great disruption of the participants' lives. A multi-party multimedia application usually requires the establishment of many real-time channels, even if each one of them is a multicast channel. If one or more of those channels cannot be reserved in advance for the starting time and the duration specified by the client, the client would certainly appreciate being informed by the service provider about other values of the starting time and/or of the duration that would make it possible to set up all the channels requested.

One way the provider could encourage advance reservations is to offer lower charges for an advance channel than for the equivalent immediate channel. These discounts could be justified with the same arguments that are the basis of similar discounts for airline tickets, i.e., easier and more effective planning.

Thus, to summarize, an advance real-time channel will be requested by specifying, besides the parameters that define an immediate channel, the following two quantities:

**(i)** the starting time, and

**(ii)** the duration.

These two times may have to be (or to be transformed into) integral multiples of a *time granule*, and the starting time may have to satisfy the constraints (if any) on advance notice, as mentioned above. In the case of a rejection of the request, the client should be notified of the reason for the rejection, and of what changes to which parameters, including (i) and (ii) above, would be effective in getting the request accepted.

## 4.3 A distributed advance reservations mechanism

A key decision concerns the organization of our advance reservation service. A natural choice in this area is the centralized one. Most of the advance reservation services in other fields are centralized (e.g., hotel rooms, meeting facilities), or at least make use of a single database (e.g., theater or airplane seats). A centralized solution for a real-time network running the Tenet protocols is feasible, but would suffer from the problems usually associated with centralization: the creation of a performance and reliability bottleneck, poor scalability, and the need to keep in the central reservation agent an up-to-date view of the present and future resource allocations throughout the network. The last problem could be solved by centralizing all channel setups, including those of the immediate channels; however, this would be a major departure from the Tenet approach, which, being targeted to large internetworks, has always tried to maximize distribution of control operations. We have therefore adopted a distributed procedure also for the establishment of advance channels, which we now describe.

In a distributed approach, the advance reservation information must be stored in the servers[2] of the network: each server has to keep track of how much of each of its resources has been reserved at various future times, besides knowing how much of each resource is set aside for those channels that already exist at the present time. This increase in the amount of state information to be recorded in each server certainly makes fault recovery more complicated and time-consuming; however, this important problem is outside the scope of this dissertation.

We divide the future-time axis of a server into *intervals* characterized by the following two properties:

**(i)** an interval does not include any instant at which a channel traversing the server starts or ends its life; these events delimit intervals but never occur within them;

**(ii)** the allocations of resources to the server's partition are constant throughout an interval; they can only change (i.e., the boundaries for some of the resources in a server can only be moved) at the transition point from an interval to the next.

The basic mechanism used to manage the resources in a server partition is the *interval table*, which lists all the channels that will traverse the server during a future interval, together with the requirements for each of the server's resources (cf. Figure 4.1). The interval table, an example of which is shown in Table 4.1, includes also the amounts of each resource that are available to the partition during the interval, as well as the totals that have been allocated to channels.

[2]As described before, a server is a network node or link.

| Channel id | Buffer space | Processing power |
|---|---|---|
| 312 | 14 | 800 |
| 174 | 8 | 144 |
| 586 | 11 | 650 |
| Resources allocated | 33 | 1594 |
| Resources available | 50 | 2000 |
| Start time | 002041735 | |
| End time | 002641735 | |

Table 4.1: An example interval table in a server

Table 2 assumes that the scheduling discipline the server implements is one that requires only buffer space and processing power to be considered as resources (if a deadline-based discipline is used, we need to consider also the "schedulability" or "delay" resource [45]). In the table, buffer space is expressed as a number of packet-sized buffers, and processing power in kbits/s; times are measured in milliseconds. We have omitted several columns that contain local bounds and other channel parameters.

When the partition in a server is empty, there is only one interval table; its top row is empty, its start time is *inception*, and its end time is *eternity*. When an advance channel request is received from a client, the source sends out an advance establishment message containing, together with all the usual traffic and QoS parameters, the start and end times of the reservation.

The arrival of this message at an empty server causes the only existing interval to be subdivided into three intervals: (inception, start), (start, end), and (end, eternity). For each interval, the corresponding interval table is created; the first and the third have the top rows empty, whereas the second has just the requested channel in it (assuming the available resources are sufficient to accept the channel, i.e., assuming that the request passes all the tests against the available resources).

The situation remains as described until a message relating to the same channel comes back from the destination(s), assuming, for simplicity of description, that no other establishment request is received by the server before this time. If the returning message is a *channel-accept* one (i.e., at least one destination has accepted the request), then the reservation is confirmed; only some of the values in the second table are modified to adjust the reservations and set the local bounds. If, on the other hand, the returning message is a *channel-reject* one, then the three tables are re-merged into the initial empty table.

This procedure is repeated at the arrival of every successive request at the server. In general, such an arrival will find the future-time axis of the server subdivided into $n$ intervals, and its expected lifetime will cover completely a fraction of them, but its birth and death may split up to two of the existing intervals; for example, in Figure 4.1, tables $T_{01}$ and $T_{56}$ will not be affected by the addition of the new channel, while $T_{12}$ will be relabeled $T_{11'}$ (its end time will change from $t_2$ to $t_{1'}$) and $T_{45}$ will be renamed $T_{4'5}$ (its start time will become $t_{4'}$ instead of $t_4$); $T_{23}$ and $T_{34}$ will be updated by the simple addition of a row corresponding to the new channel, and $T_{1'2}$ and $T_{44'}$ will be created from $T_{12}$ and $T_{45}$, respectively, in the obvious way.

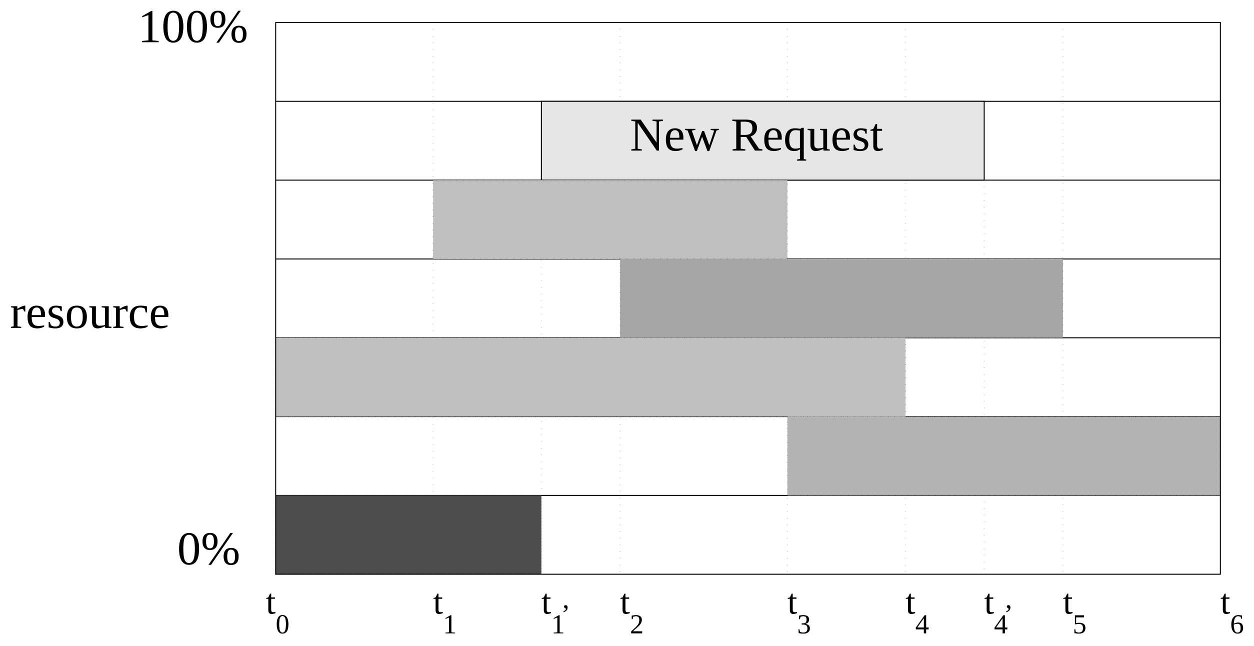

Figure 4.1: Effects on the intervals and interval tables of the addition of an advance channel

Thus, after the arrival of the new request, the server will have two more interval tables; to put a curb on the proliferation of tables, we use the time granules that have been mentioned in Section 4.2 and earlier in this section, with the provision that a client-specified time not satisfying this rule will be modified to coincide with that of the nearer inter-granule transition. Of course, if the return message is a *channel-reject* one, the new interval tables (e.g., $T_{1'2}$ and $T_{44'}$) will be deleted, and the others restored to their previous state.

When the current time becomes equal to the start time of an interval, the table of the previous interval is deleted[3], and the table of the next interval becomes the current table. Those advance channels whose start time coincides with the start time of the current interval, i.e., with the current time, can spring to life automatically in all the servers they traverse without any need for establishment, thereby producing the illusion of being connectionless, while in reality they were established in advance. This result can be smoothly achieved in networks whose nodes have clocks that are kept in approximate synchrony by, for instance, protocols such as NTP. Note that the intervals have variable lengths so as to minimize the number of tables in a server. In fact, this number at any time is bounded from above by twice the number of advance channels established in the server at that time.

[3]And the "initial" interval extended to include the time covered by the "previous" interval

## 4.4 Discussion

In the previous section, we described the interval-table-based mechanism for providing advance reservations for real-time channels. In this section, we will discuss several important system issues that concern these advance reservation mechanisms. We first talk about the client-service interface: the set of services that can be offered to the clients and the alternatives that we decided to support in our initial implementation. We will then talk about the constraints that the network service provider may impose on the advance notice period as well as on the time granularity; also how placing such limits can be useful to the network. We will then discuss how the resource partitioning mechanisms can be used to provide a more efficient and effective advance reservation service; we will conclude this section with a discussion of aggregation, by which the admission tests for multiple servers in a subnet can be run at the same physical node.

### 4.4.1 Service interface

As we discussed in Section 4.2, when advance reservations are introduced in a real-time communication service, the service provider has to do some planning for the future allocation of resources, and this planning would be easier if the expected durations of the channels were known. This determination may be made by either (a) requiring the clients to explicitly specify the start time and the duration for the advance-reserved channel, or (b) estimating the call duration ("holding time") based on long-term traffic analysis. For our advance reservations service, we chose the first approach, mainly because we could not rely on the estimates for the call holding time in a guaranteed-performance environment.

An important interface issue concerns the set of services offered. The resource reservations can be classified on two orthogonal criteria: (a) immediate vs. advance reservations, and (b) whether the reservation is for a limited or unlimited time duration. This leads to four possible sets of reservation types: (I) *immediate* channels with *unlimited* duration, (II) *immediate* channels with *limited* duration, (III) *advance* channels with *unlimited* duration, and (IV) *advance* channels with *limited* duration. The mechanisms described in Section 4.3 support all four options; for immediate channels, the start time should be set to the current time, while, for unlimited duration channels, the end time is set to *eternity*. It is a policy issue as to whether a particular network will provide only a subset of the above services. For example, the current Suite 2 implementation, as described in Chapter 5 only permits the clients to specify options (I) and (IV), though option (II) is available as a network-provided option; if the client makes a request for an immediate channel with unlimited duration, and if the requested resources are only available for a limited time, the network can offer the client this option. The client, of course, is free to reject this offer.

The network activities for setting up real-time channels include admission control and resource reservation (RCAP), as well as setting up state information for data forwarding in the switches (RTIP). Admission control is done at reservation request time, but the RTIP state set-up should be done at the start of channel lifetime. For setting up this state information, there are two alternatives: (a) at the start of data transmission time, RCAP sends a message to set up the desired state, or (b) the client sends another message ("state set-up") before starting data transmission; this messages causes the establishment of the desired state at the servers along the channel path. We chose option (a), mainly because it preserves uniformity between advance and immediate channels.

### 4.4.2 Constraints on advance notice period

The mechanisms described in Section 4.3 support reservation requests with arbitrarily large (or small) advance notice, as well as any "time-granularity". On the other hand, in Section 4.2, we stated that the clients would probably accept reasonable limits on the minimum and the maximum advance notice with which reservation requests can be submitted if such limits were imposed by the provider. We now present several reasons why, as a policy matter, the network service provider may impose these limits:

- The service charges may be determinable for a reasonable time duration only; the network cannot reasonably commit the service rates for channels that will be used ten years from now. Due to this reason, the network should limit the maximum advance notice that a reservation request may have.
- If the service provider offers lower charges for advance channels than for equivalent immediate channels, then it is important that the clients not be able to cheat the service provider by providing very little advance notice for their reservation requests (for example, a few seconds).
- By placing reasonable limits on advance notice period as well as on time granularity, the network can limit the proliferation of interval tables; this reduces both the computational overhead and the storage space required for maintaining the interval tables.

### 4.4.3 Interactions with resource partitioning

In our real-time communication service, immediate channels co-exist with advance channels; this co-existence leads to the following two concerns:

- Acceptance of advance-reserved channels in a server can lead to proliferation of *time intervals* in the interval table; also, the immediate channels (with indefinite duration) span the time intervals. The combination implies that accepting advance channels and immediate channels in the same server may lead to a significant increase in the computational overhead for running admission tests for immediate channels. For example, with 40 channels accepted in advance, the interval table may contain up to 80 time intervals; this implies that, for an immediate channel, the computational overhead would be about 80 times higher than if no advance channels had been accepted. Higher computational overhead implies higher establishment latency, which is especially undesirable for immediate channels.

- An added concern is that the advance requests may starve out immediate channels (or vice versa). For real-time communication services to achieve widespread usage, it is important that the network managers be allowed to control the services effectively; this implies that the network service managers be able to control the amount of resources that the advance requests may be able to reserve.

The resource partitioning mechanisms described in Chapter 3 can address the issues raised here in the following manner: we can partition the resources at a server (to divide the one physical server into two virtual sub-servers) and use one partition for supporting advance channels and the other for supporting immediate channels. With this set-up, the admission test computation for immediate channels is not affected by the advance reserved channels; also, the managers can control the total resources that can be reserved in advance. The dynamic partition allocation change mechanisms (described in Section 3.4) allow the network managers to control the "borrowing" of resources between immediate and advance partitions. Also, if the immediate partition is full, the network can "move" the channel to the advance partition, thereby providing the option III described in Section 4.4.1

### 4.4.4 Interactions with routing

Advance reservations have an interesting implication for the routing system. If the routing decisions are dynamic and load-dependent, then for an advance reserved channel, these decisions should take into account the network load during the channel's lifetime. At channel establishment time, the routing system does not have full information about the network load during the channel's lifetime. At best, the network knows about some other advance reserved channels[4] overlap with that of the current request; the routing system should use this information to make the routing decisions. This is the mechanism that we proposed for the Tenet Suite 2 implementation [7].

On the other hand, advance reservations make rerouting channels much easier. If a channel fails admission tests at a particular server, the network can "reroute" that channel, or other advance-reserved channels holding reservations at that server, around that server before data is transmitted on that channel. This "rerouting in advance" is much easier, as the network has a lot more flexibility. The network does not have to worry about mis-ordered and/or duplicate packets in this case.

### 4.4.5 Aggregating admission test computations

As we mentioned in Section 3.4, admission control can be viewed as an accounting system that ensures that the network has enough resources to support the various requests for guaranteed-performance connections. With resource partitioning, we can separate this accounting and bookkeeping for immediate and advance channels. In Chapter 1, we described a fully-distributed technique for establishing channels and for running admission tests. A key motivation[5] for a fully-distributed approach for setting up *immediate* channels

[4]These are the channels that have already been requested.

[5]This is in addition to all advantages of having a fully-distributed mechanism.

is that the system has to set up state along the channel path anyway; we cannot save much by more centralized decision-making. On the other hand, for advance-reserved channels, the network can gain by "aggregating" the decision-making. For example, a server can "assign" its admission control decision-making to a remote node. In a sub-network, all nodes can assign the admission control decision-making for all advance channels to a single node, thereby choosing more localized/centralized decision-making.

This localization can be very useful:

- It reduces the establishment latency, as the admission tests for all servers in this sub-network are made at a single node, without requiring multiple inter-node messages. The results in [7, 59] show that inter-node messages dominate connection set-up latency.
- Fault-handling becomes more difficult in advance reservation systems; a key concern is that the advance reservation information be saved in stable storage. With a centralized approach, the information can be put in stable storage at this particular node; it is more difficult to ensure that information is stored on stable storage in a fully distributed system (distributed consensus is required in this case).
- Rerouting, as described in Section 4.4.4, becomes easier if all resource reservation information is available at a single node.

Thus, the network can gain by aggregating the resource reservation decision-making, accounting and book-keeping in a pre-selected set of network nodes.

## 4.5 A simulation-based evaluation

We performed a number of simulation experiments to evaluate the performance of our advance reservation mechanisms. For this discussion, we have selected four interesting sets of simulation experiments. In the first set, we ran simulations of unicast channel requests, with and without the advance reservation mechanisms. To distinguish the effects of the two components of our advance reservation mechanisms, (namely partitioning-induced protection and priority changing) which we describe below, we repeated the previous experiments with workloads that eliminated the effect of one of these two components. The third set of experiments considered multi-party advance reservation requests that varied in conference size and in advance notice period, while the fourth set evaluated the effect of time granularity on the performance of our advance reservation mechanism. In this section, we describe the simulation scenario, the workload, and the results obtained with these experiments.

### 4.5.1 Simulator description

We ran these simulations on an enhanced version of *Galileo* [72], an object-oriented real-time network simulator. This version provides complete support for multi-party real-time communication protocols, including support for advance reservations.

Our goal was to make the experiments as realistic as possible, so that we could confidently predict the behavior of our implementation of the advance reservation service in the Tenet Suite 2 [60]. For example, we used the NSFNET backbone network topology in our simulations (see Figure 2.8). We assumed 45Mbps for each link, and we set the propagation delay along the diameter to 40 ms[6] . In all our experiments, we created a partition for non-real-time traffic containing 20% of the resources in each server; this left up to 80% of the resources for real-time communication. We also made the amount of buffer space in each server large enough that the buffer space test would always be successful, and we chose the delay bounds large enough so that schedulability would never be a problem (for the Earliest-Due-Date (EDD) scheduling discipline that we used in our simulations). Thus, bandwidth or processing power was the limiting resource in all servers and all scenarios.

### 4.5.2 Simulation workload and evaluation metrics

In all the experiments, the sources and destinations for the channels were uniformly and independently distributed among the network nodes. We ran the same simulations without advance reservations and with advance reservations for different resource allocations to the two partitions.

To keep comparisons meaningful, we only considered relatively homogeneous workloads, where all channels have identical traffic descriptions, and all destinations specify identical performance requirements:

- Deterministic delay bound $D = 400$ ms ;
- Deterministic jitter bound $J = 16$ ms;
- Minimum inter-packet time $Xmin = 8$ ms;
- Maximum packet size = 8 Kbits.

The traffic description corresponds to that of a compressed video stream at a peak rate of 1 Mbps, at 30 frames per second, with four data packets per frame ; the average rate did not matter, because bandwidth was allocated according to the peak rate, and bounds were deterministic. The start times and the durations of the channels varied randomly and uniformly within specified time intervals.

We performed many sets of experiments with varying workload parameters; for each of these workloads, we ran many simulation experiments, and averaged the results thus obtained.

The main metric we adopted for evaluation and comparison was the acceptance ratio, defined as follows:

$$\text{Acceptance ratio} = \frac{\text{Number of destinations reached with advance reservation}}{\text{Number of destinations reached without advance reservation}}.$$

[6]The simulation workload and evaluation metrics are the same as those for the simulation experiments for resource sharing and resource partitioning, in Section 2.5 and Section 3.3 respectively; this information is provided here only for completeness.

We were also interested in the timeliness and the computational cost of channel establishment, for which we used a different metric: the computational overhead associated with admission control. In the simulations, we use the admission control tests for the EDD scheduling discipline [49]. The time required to run these tests at a given node is proportional to $n$, the number of already accepted resource allocations at that server when a new request arrives. The overall establishment overhead is computed as the sum of these $n$ (already accepted allocations) at each node along the route of the channel. This component of admission control computations increases as the real-time network load (number of accepted channels) goes up. Computational overhead for other activities (for example, for collecting the state information) does not change when this load increases (though it depends on the route selected). We therefore focus on this admission test computation overhead for performance evaluation. This led us to the following metric for comparing computational overheads:

$$\text{Overhead ratio} = \frac{\text{Total computational overhead with advance reservation}}{\text{Total computational overhead without advance reservation}}.$$

### 4.5.3 Simulation experiments

#### First set: Simple workload

In the first set of experiments, we compared the following two scenarios:

- *without advance reservations*: 150 simplex unicast connections, and 50 10-person conferences (each conference requiring 10 9-destination multicast channels), all in the same partition, which was allocated 80% of the network's bandwidth; and
- *advance reservations*: two partitions, with 150 simplex unicast connections in the immediate partition, and 50 10-person conferences in the advance partition; with varying partition allocations, so that the total resource allocation for these partitions equals 80% of the network's bandwidth.

In these (and subsequent) experiments, the channel duration varied uniformly between two and three hours for conference channels, and between 30 minutes and ten hours for unicast channels. The advance notice period varied uniformly between three and four hours (in the third set of experiments, the advance notice period varied from six to seven hours for large conference). The reservation request arrival time was chosen from a uniform, random distribution over about ten hours. We deliberately chose this workload to saturate the network, because we wanted to observe the network's behavior under heavy real-time load. It should also be observed that, in this workload, all conferences are of the same size, and that there exist resource sharing relationships among the conference channels (only up to 2 of the 10 channels constituting a conference may be active at any given time) so that they can share resource allocations [62, 63].

In Figures 4.2-4.9, we report on the horizontal axis the fraction $f$ of the total resources that is allocated to the immediate partition. The advance partition's allocation is $100(0.8 - f)\%$ of the total resources. Note that the boundary between the partitions remained fixed for the duration of each simulation run.

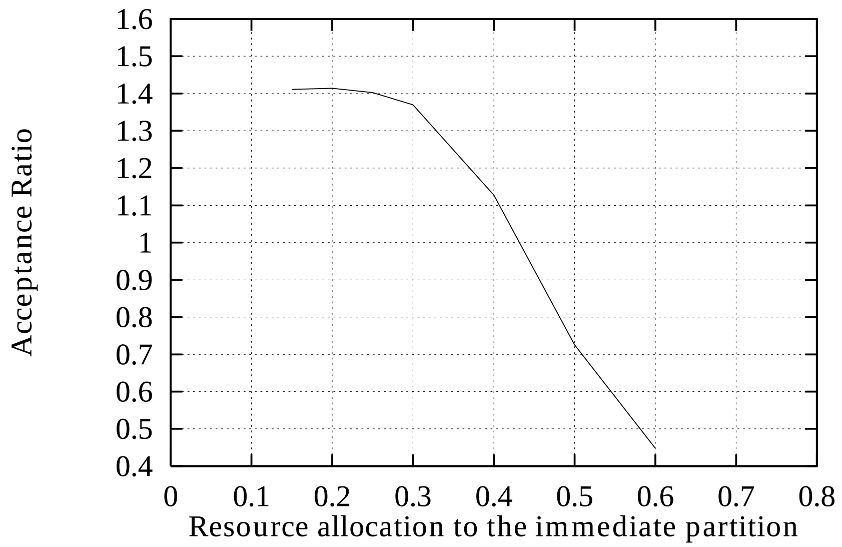

Figure 4.2: Acceptance ratio for the first set of experiments (simple workload)

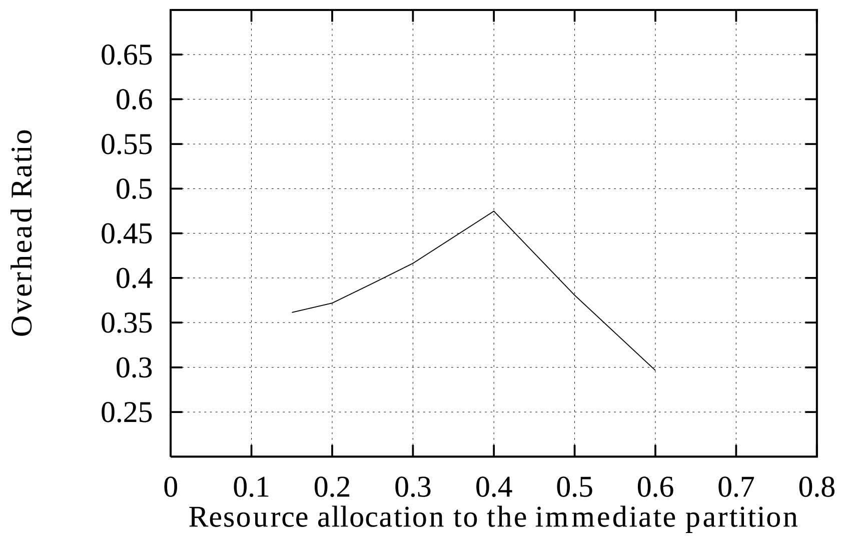

Figure 4.3: Overhead ratio for the first set of experiments (simple workload)

In Figure 4.2, we observe that there is a large region in which the acceptance ratio is higher than 1, i.e., in which the acceptance rate is higher with advance reservation mechanisms than without these mechanisms. However, there is also a large region in which the ratio is lower than 1. Thus, moving the resource allocation boundary is necessary whenever the workload is such that the boundary falls in an area with low acceptance ratio.

Figure 4.3 shows that our advance reservation mechanisms reduce the computational overhead of admission control for all allocations. As we increase the allocation to the immediate partition, the overhead increases (due to resource sharing, the computational overhead is higher for single channels than for channels that belong to a conference [62]; the overhead increases because we accept more individual channels at the expense of conference channels). However, after reaching a peak value, the overhead starts decreasing (the overhead reduction due to the increasing rejection rate of conference channels starts dominating and offsetting the overhead increase due to the growing acceptance of individual channels).

### Second set: Isolating the effects of two components

Our advance reservation mechanisms affect resource reservations in two ways: first, resource partitioning provides isolation and protection between the advance requests and the immediate requests; second, advance reservation provides higher priority to connections that come with larger advance-notice periods. In the second set of experiments, we ran additional simulations to separate the effects of the resource partitioning mechanisms from the effects of priority changes. For this, we added the following scenario to those considered in the previous set of experiments:

- *only partitioning*: two partitions, with 150 simplex unicast connections in the first partition, and 50 10-person conferences in the second partition, where resources are not reserved in advance;

As the graphs of Figure 4.4 show, the gains (and losses) observed in Figure 4.2 arise primarily from the isolation and protection provided by the resource partitioning mechanisms, since the two curves ("advance reservations" and "resource partitioning") essentially coincide. However, we should remember that, in our simulations, all conferences were of the same size (10 members each); it would be more realistic to simulate different-sized conferences, as we did in the third set of experiments discussed below. As shown in the second diagram in Figure 4.4, the overhead of admission control is substantially lower in the *only partitioning* case than in the *advance reservations* case, where multiple tables are manipulated.

### Third set: Multiple conference sizes and advance notice periods

In the third set of experiments, we considered different types of conferences, under the assumption that larger conferences tend to be requested with larger advance notice periods. To reduce the simulation time, we decided to leave the immediate partition empty; this did not affect our results because we were interested only in the effects of the priority changes on the channels expected to be usually reserved in advance.

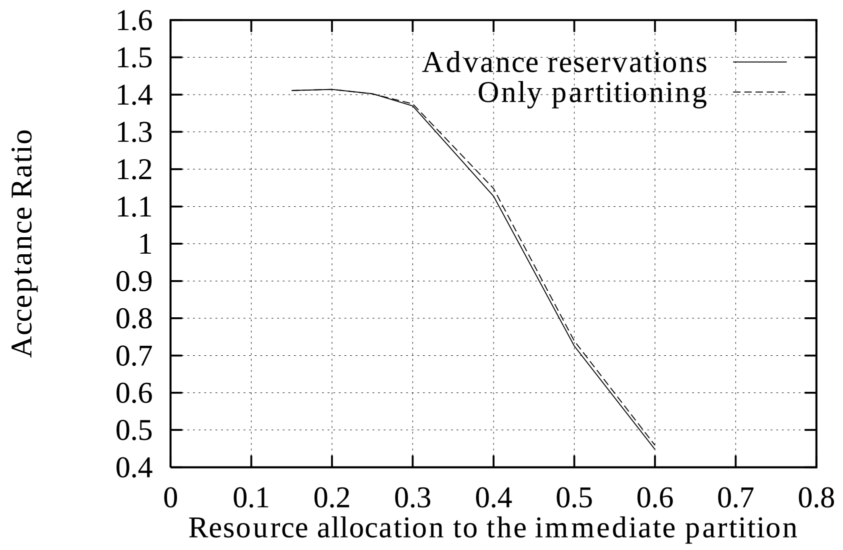

Figure 4.4: Isolating the effects of the two components: acceptance ratio

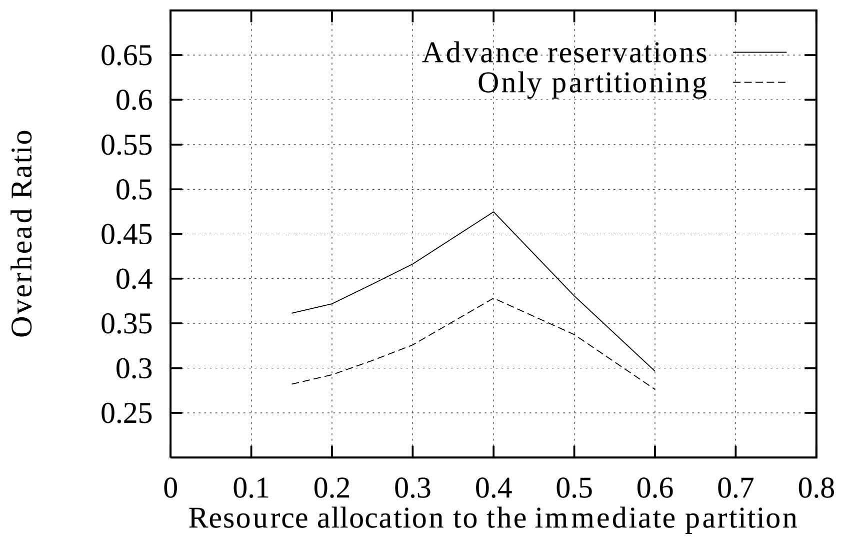

Figure 4.5: Isolating the effects of the two components: overhead ratio

We compared the following three scenarios:

- 10 50-person conferences, and 50 10-person conferences, all in the same partition, which was allocated 80% of the network's bandwidth; we call this the *without advance reservations* case;
- two partitions: the first partition empty; 10 50-person conferences and 50 10-person conferences in the second partition, where resources are not reserved in advance; we call this the *only partitioning* case; and
- two partitions: the first partition empty; 10 50-person conferences and 50 10-person conferences in the second partition, where resources are reserved in advance; we call this the *advance reservations* case.

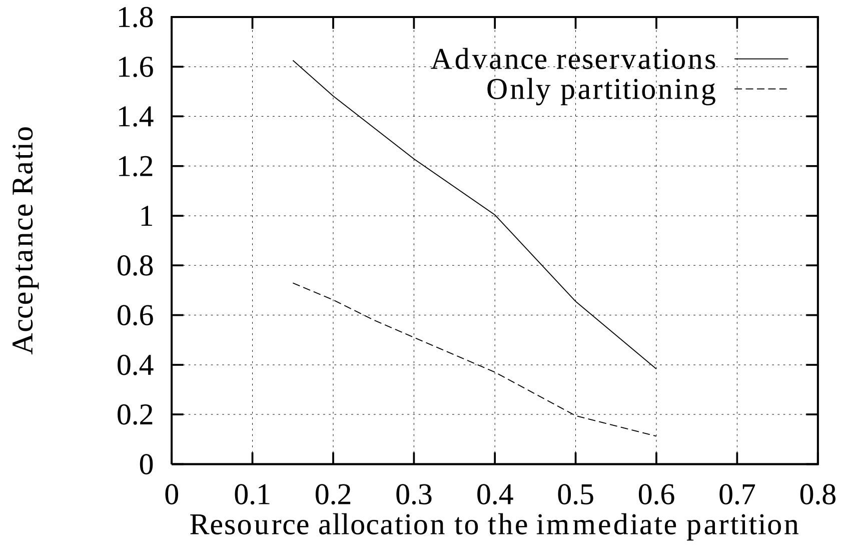

Figure 4.6: Multiple advance notice periods and conference sizes: acceptance ratio

As the graphs in Figure 4.6 show, when we consider conferences with different sizes, the higher priority provided by the advance reservation mechanisms leads to a considerable increase in the acceptance ratio. Thus, the experiments show that, in general, both components of our advance reservation mechanisms help improve the connection acceptance rate. The overhead ratio also increases, due to the higher acceptance ratio, and, because of the absence of immediate requests, does not have a maximum for an intermediate value of the relative resource allocation.

### Fourth set: Effect of time granularity

In the fourth set of experiments, we decided to observe the effects of *time granularity* in advance reservations.

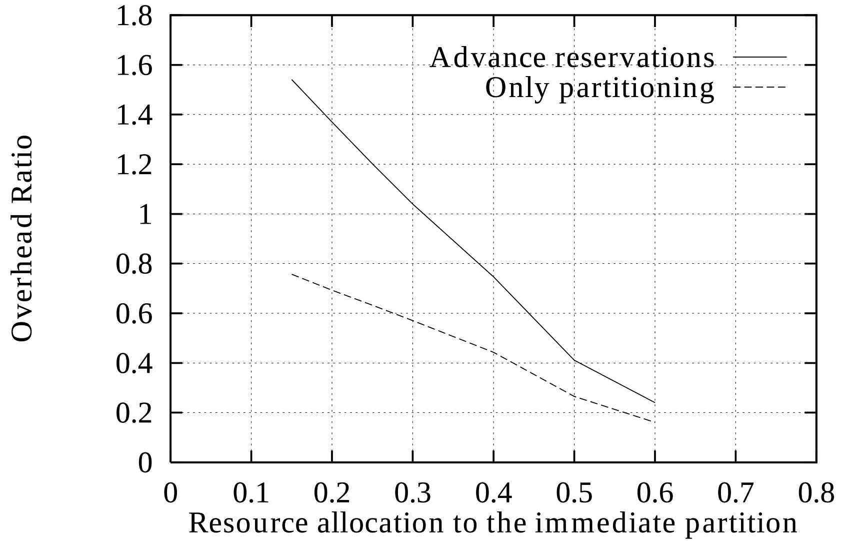

Figure 4.7: Multiple advance notice periods and conference sizes: overhead ratio

We ran our tests on the *advance reservations* and *without advance reservations* scenarios of the third set while varying the granularity of start times and durations from 1 minute to 15 minutes. We assumed that the users would ask for connections with start times and durations that vary randomly and uniformly within specified time intervals.

As in all of the experiments described above, we created a partition for non-real-time traffic containing 20% of the resources in each server. Figures 4.8 and 4.8 show that the acceptance ratio and the overhead ratio decrease slightly as we increase the time granularity. However, the lower acceptance ratio may be due to the assumption made in the simulations that the start times are completely random within a small time interval. So, we conclude that time granularity does not appreciably affect the performance of our mechanisms.

### 4.5.4 Resource allocation gains

With advance reservations, we obtain resource allocation gains from two distinct sources: the protection provided by resource partitioning, and the priority change provided by the advance reservation mechanisms. The fundamental source of resource allocation gains is *resource sharing*; due to resource sharing, the conference channels are more efficient in allocating resources. These two factors merely enhance/facilitate this extra efficiency. This increase (in allocation efficiency) occurs because these factors cause the admission system to favor the conference channels, at the expense of non-conference channels. For example, in the second set of experiments, without advance reservations, the system rejected about 12% of the unicast channels and one or more destinations of about 91% of the conference channels (about 51% of conference destinations not reached); under advance reservations, with the advance partition allocated 40% of the resources and the immediate partition allocated 40% of the resources, the system rejected about 25% of the unicast channels and

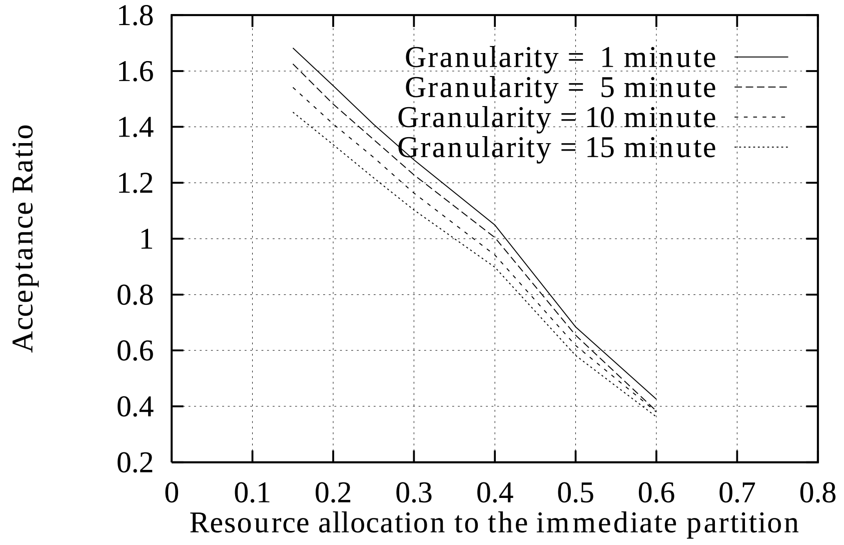

Figure 4.8: Effect of time granularity: acceptance ratio

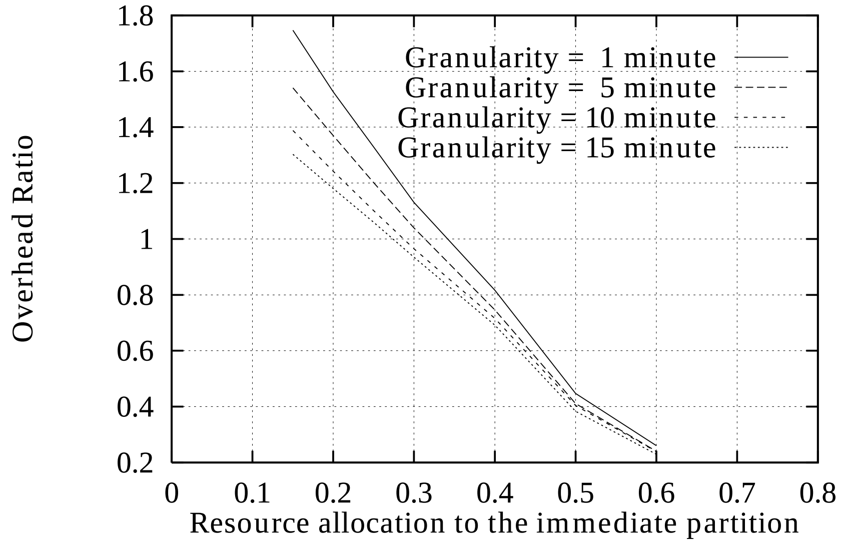

Figure 4.9: Effect of time granularity: overhead ratio

one or more destinations of about 20% of the conference channels (about 18% of conference destinations not reached). If there were no conference channels, then we would observe no extra allocation gains with advance reservations. However, if the system has a workload mix that has either a mixture of conference and non-conference channels, or conferences with different sizes and different advance notice periods, we expect results similar to those obtained in these simulations.

In our fully-distributed advance reservation service, the allocation gains at a server depend only on the *real-time* load at that server. These allocation gains depend on the network topology, the routing system, the session holding times, the request arrival patterns, and other such factors only because these factors influence the real-time load. If these factors change but the real-time load remains the same, then the results should remain the same. For example, simulations of Chapter 2 show that the allocation gains are similar for networks with very different topologies.

## 4.6 Summary

In this chapter, we presented a fully-distributed scheme for advance reservations of real-time connections. We also presented the results of our simulation experiments; these verified the usefulness of our techniques. These experiments show that our distributed mechanisms work, and their cost is affordable. An interesting feature of our advance reservation mechanisms is that they favor channels that belong to conferences (and, because larger conferences usually have larger advance notice periods, our mechanisms favor larger conferences over smaller conferences). Conferences may not be held if there are no advance reservations; for example, a conference may not be held at all if all its channels, or a substantial fraction of them, are not established.

In the last three chapters, we have described resource sharing, resource partitioning, and advance reservations; these three constitute the core of our multi-party real-time communication system. In the next chapter, we will describe how we put these mechanisms together in Suite 2, our multi-party real-time protocol suite.

# Chapter 5

# Implementation: Suite 2

In the previous three chapters, we discussed the three key components of our multi-party real-time communication research: *resource sharing*, *resource partitioning*, and *advance reservations*. Resource sharing mechanisms provide efficient support for large-scale conferences; resource partitioning provides an effective tool for the network managers to control and distribute resource allocation among the various network service clients; advance reservations enhance the service usability, and at the same time, can help improve the network performance with better planning and off-line routing potential.

As we mentioned before, it is critical that these components work well together, and with other components of our real-time communication system (e.g., routing); over the last four years, we have designed the Tenet Protocol Scheme 2 and implemented the Scheme 2 algorithms and techniques in the Tenet Protocol Suite 2. These protocols incorporate the ideas discussed in the previous chapters to provide network support for multi-party real-time communication. In this chapter, we describe these Tenet protocols. For the sake of brevity, we will not distinguish between *Scheme 2* and *Suite 2*; the former term refers to the algorithms and techniques designed, while the latter refers to the incorporation of these mechanisms in a set of protocols. We will focus our attention on the resource sharing, partitioning, and advance reservations related aspects of the design and implementation; we will only briefly outline the other components.

In designing this second generation of Tenet protocols, we tried to minimize the changes from the previous generation (unicast) Tenet protocols. We kept the same separation of real-time data delivery and control protocols between RMTP/RTIP and RCAP (as described in Chapter 1 and illustrated in Figure 1.1); within this overall framework, we completely re-designed the signaling protocol (RCAP) for supporting multi-party communication. We briefly describe the new Scheme 2 RCAP in Section 5.1. We will illustrate this design with a simple connection establishment example in Section 5.1.5. Supporting multi-party communication required two changes in RTIP: for multicasting, and for resource sharing; we discuss these topics in Section 5.2.

## 5.1 Suite 2 RCAP

In this section, we describe RCAP, the signaling protocol for multi-party real-time communication. In the unicast protocols, RCAP's primary tasks were connection establishment and teardown; the connection establishment also included mechanisms for rendezvous between the sender and the receiver. In multi-party communication, these primary tasks become more difficult because of the multiplicity of senders and receivers; in addition to previously described issues as service usability and network management and control, we also face critical concerns in security and access control, scalability, and reliability. The signaling protocol occupies the center stage where we address these key concerns.

In this section, we first describe the key design goals and trade-offs in this design; we will then describe the RCAP design and its embodiment in the Tenet Suite 2. We will also illustrate the interaction of various components and modules with a simple connection set-up example. We will conclude this section with some preliminary performance measurements over our prototype implementation.

### 5.1.1 The mechanisms

In Chapter 2, we described the resource sharing mechanisms; we also described resource partitioning and advance reservation mechanisms in Chapter 3 and Chapter 4 respectively. In this subsection, we will briefly review these mechanisms in the Tenet Suite 2 framework.

The resource sharing mechanisms affect RCAP as well as RTIP; for RCAP, resource sharing requires two sets of changes: (a) RCAP is now required to maintain resource sharing information, i.e., the different *resource sharing* groups, their respective aggregate traffic specification(s), and the member channels that belong to each group; and (b) during channel set-up at any RCAP daemon, the daemon must determine if resource sharing can be used to reduce the resource allocation at that node. If the RCAP daemon uses resource sharing, RTIP requires enhancements to support traffic policing and rate control for these resource sharing channels on the group aggregate traffic specification.

The resource partitioning and advance reservation mechanisms do not affect RTIP at all; they only require changes in RCAP. Supporting resource partitioning requires changes to admission control tests (as described in Chapter 3) as well as management software for creating new partitions and for setting/changing the resource allocations to these partitions. As described in Chapter 4, the basic mechanism for supporting advance reservations is the *interval table*, which lists all the advance channels that will traverse the server during a future interval, together with their requirements for each of the server's resources. The interval table includes also the amounts of each resource that are available to the advance partition during the interval, as well as the totals that have been allocated to advance channels.

### 5.1.2 Suite 2 RCAP design

RCAP design goals included the following, in addition to the design goals and objectives described in the previous chapters (for devising mechanisms for resource sharing, partitioning and advance reservations):

- **Modifiable and extensible**: An important design consideration was that the protocols be easily modifiable and extensible, for example, to incorporate new traffic specification models, packet scheduling disciplines, failure-handling techniques, underlying network technologies etc. To achieve this modifiability, we decided to minimize the set of assumptions that we make about the underlying network; we also spent considerable time and effort in designing the interfaces between modules to ensure this extensibility.
- **Speed**: Although we do not need hard-real-time guarantees on RCAP requests (e.g., RCAP will establish connectivity within 50 ms), it is desirable that the connection establishment be fast; most real-time clients cannot afford long waits for their transmission to start [41]. These factors led to our single round-trip connection establishment procedure; we described this procedure for unicast channels in Chapter 1; we will describe the corresponding establishment procedure for multi-party channels in Section 5.1.5.
- **Ease of use, management, and implementation**: It is important that the services provided should be "easy-to-use" for the clients; it is also important that the mechanisms designed should permit the network managers to effectively control the network, and that the mechanisms be easy to implement. These factors are critical for our research effort, because we can learn a lot from using (and running experiments on) a prototype implementation.
- **Location independence**: In our model, a typical conference may be organized by a party (the organizer) who may not otherwise participate in the conference. This implies that a channel may be set up (or torn down) by a user who is not located at either the source or any of the destination(s) for that channel. Thus, RCAP must support the users making the channel set-up/modification/teardown calls from anywhere in the network, and all the network's behavior should be independent of where the calling user is located.

These considerations led to our object-oriented design for RCAP; we now enumerate some salient components and features of this design:

- **Partial establishment semantics**: Multicasting raises an important concern regarding connection establishment semantics: a channel establishment request may only be able to obtain resources along the paths to a subset of destinations. At least two alternatives exist: (a) the network can declare that the connection establishment succeeded (partially), or (b) the network can declare that the connection establishment failed (*all-or-nothing*).

The first option simplifies the implementation and provides "best-effort" connectivity; the second option provides stronger semantics (if the connection establishment succeeds, the sender knows that all destinations' performance requirements are met). We chose the first option, mainly due to ease of implementation and scalability considerations.

- **Location and naming service**: Location independence requires that clients be able to set-up and/or manipulate the channels from anywhere. This implies that all network entities (including multicast groups, channels, entities for resource sharing, and so on) be addressable and accessible from anywhere in the network; these entities must have globally unique identifiers ("names"), and the naming service should be able to return the location of the node where that entity resides[1].

  For our prototype implementation, we chose a simple naming convention to address this issue; all names are 64-bit long, where the first 32 bits provide the location address; this greatly simplifies the location and naming system (LNS), though it also implies that these entities can not migrate between nodes in the network.

- **Ranges for performance parameters** The Tenet Scheme 2 allows the users to specify ranges of performance parameters (instead of single values); for example, the user specifies a desired value for end-to-end delay bound as well as upper limit on acceptable values for the delay bound. If the network can not meet the desired service but can provide service within the acceptable range, the channel establishment succeeds. In many cases, the use of ranges avoids repeated re-negotiations between the network and the client [112, 60, 99, 100].

### 5.1.3 Client interface

Suite 2 RCAP is an object-oriented distributed system for supporting multi-party real-time applications. We now describe how the three components – resource sharing, resource partitioning, and advance reservations – affected the client interface that RCAP provides. As we described before, RCAP provides a location-independent service; the result of an RCAP request is independent of the network node (and the RCAP daemon) that received the user's request.

Logically, a channel transmits data from a single sender to a Target set; thus, to establish communication, the organizer(s) must first create a Target set[2] and then only can the organizer set up the appropriate channel(s) and any resource sharing group(s).

Resource sharing introduces a naming problem: To specify resource sharing relationships, we need to specify the channels that can share resource allocations; this requires the clients to specify the *channel-ids* for these channels. Now, in the previous-generation RCAP, these channel-ids were returned as a result of the channel establishment process; in

[1]Some of these entities can be distributed. For ease of explanation, in this discussion, we assume that these entities are not distributed.

[2]As mentioned before, the object creator gets the authorization key to access and manipulate that object; the creator can pass the key to other authorized users.

our case, we need to specify these sharing relationships (and thus, the channel-ids) before we establish the channels. To get around this problem, we separated the channel creation (and naming) from the channel establishment: the client first invokes the network to create the channel (and to return the channel-id to the caller); the client then invokes channel-establishment (specifying the channel-id in this call). In between these calls, the client can provide the resource sharing related information.

We now illustrate this client interface with a simple example. Consider a client who wishes to set up three channels, all advance channels, of which two can share resource allocations. The sequence of client requests for this case can be:

1. `Create_Target_Set()`: This call takes no parameters and returns `Target_Set_Id`, say `TS1` in this case.
2. `Create_Channel()`: This call takes the following parameters: Target_Set_Id (`TS1`), channel source node (say `A`) and the partition whose resource the channel can use; it returns `Channel_Id`, say `ChA` in this case.
3. `Create_Channel()`: Two more calls, to create `ChB` and `ChC` respectively.
4. `Set_Advance()`: This (optional) call is only required for advance-reserved channels. It takes two parameters: `Channel_Id` and the channel start and end time. In our case, the client will make three calls, once for each channel.
5. `Create_Group()`: This call takes two parameters: the aggregate traffic specification and the maximum concurrency (as described in Chapter 2). It returns `Group_Id`, say `G` in this case.
6. `Include()`: This call takes two parameters: `Channel_Id` and `Group_Id` and informs the network that the specified channel belongs to the specified group. In our case, the client will make two calls, once with parameters ( `ChA`,`G`) and the other with parameters ( `ChB`,`G`).
7. `Establish_Channel()`: This call takes one parameter, the `Channel_Id`. In our case, the client will make three calls, once for each channel.

### 5.1.4 Suite 2 RCAP architecture

Figure 5.1 shows the real-time communication system: there exists one RCAP daemon on every network node and each RCAP daemon talks with the RMTP and RTIP entities at that node. Each RCAP daemon also talks with the applications at that node, as well as with other (remote) RCAP daemons. In addition, each RCAP daemon talks with the routing system through its Routing Stub Object; in this manner, this design isolates the RCAP daemon from changes in the routing system. Any changes in the routing system are reflected only in the Routing Stub Object.

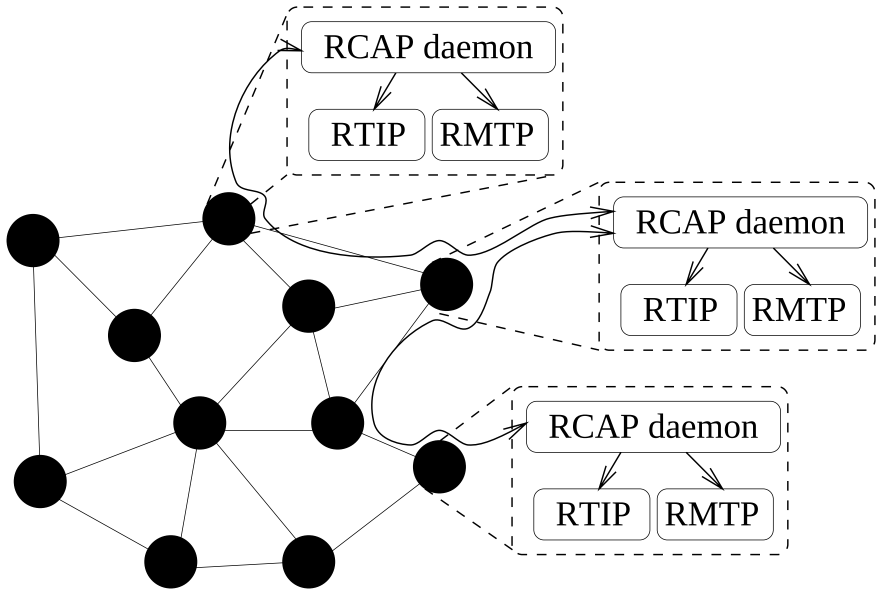

Figure 5.1: Real-time communication network with RCAP

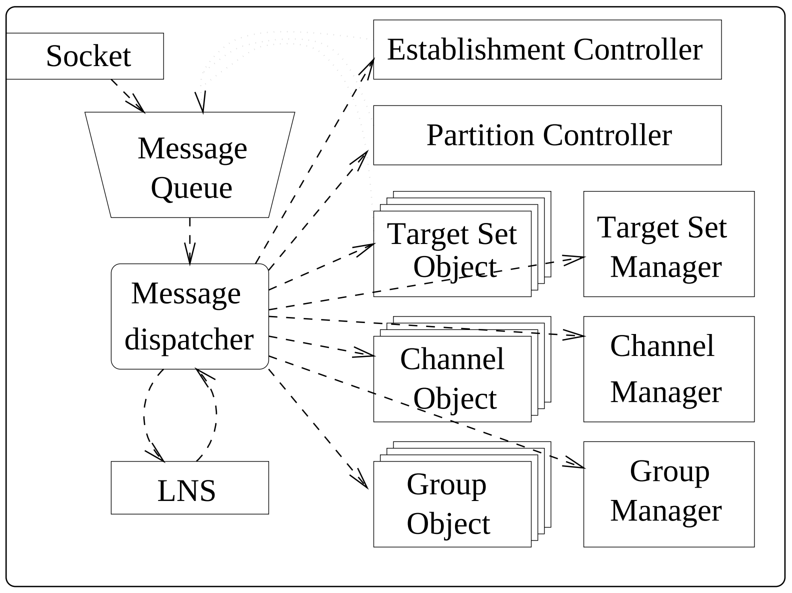

Figure 5.2: Internal structure of RCAP daemon

In this section, we will describe the internal structure of RCAP; we will describe the various components of RCAP as well as their interactions; in this description, for ease of explanation, we present a slightly simplified (and abridged) view of the internal RCAP structure and interactions. Figure 5.2 shows the internal structure of the RCAP daemon. The RCAP daemon design is object-oriented; each daemon consists of a collection of *RCAP objects* (i.e., the objects inside an RCAP daemon) along with a message dispatcher and a Location and Naming Service (LNS). For example, there exists one Target set object for each user-created Target set, one Channel object for each created channel, and one Group object for each channel group (for resource sharing). At each RCAP daemon, there also exist one manager object for each of these object classes (Target set, Channel, Group); upon user requests, these managers dynamically create (and destroy) objects of their respective classes. At each RCAP daemon, there also exist several other objects, including a Partition Controller (for managing partition allocation consistency), and an Establishment Controller (for handling channel setup and teardown requests).

In RCAP, all inter-object communication is conducted through RCAP messages (regardless of whether the objects are located on the same RCAP daemon or not); this choice greatly reduces the coding complexity and helps avoid deadlocks. Additionally, a network-generated pseudo-random key is associated with each object for security and access control; the interested reader is referred to [85] for more information about the security scheme.

When a message arrives at an RCAP daemon, the RCAP dispatcher contacts the LNS to determine if the addressed object resides on a remote node; if so, the message is forwarded to that node. Else the LNS returns a pointer to that object and the dispatcher hands over the message to that object.

The RCAP daemon may also be divided into two subsystems: one for managing multi-party communication information, including that relating to Target sets, channels, and groups; the other subsystem for managing system resource allocation, resource partitioning, channel establishment, and running admission tests. The multi-party information management system includes the following objects:

- **Target set object**: There exists one Target set object for each Target set in the system. Typically, the Target set object is located at the RCAP daemon that received the request to create the Target set. Each Target set object maintains information about the destinations (addresses and performance requirements) as well as the channels that belong to that Target set. When a user requests RCAP to establish a channel, the corresponding Channel object contacts its Target set object to obtain the destination membership list and then contacts the routing system to obtain the (multicast) route for the destinations. When a new destination joins a Target set, the Target set object contacts all established channel(s) (actually, their corresponding Channel object(s) ) to request them to extend ("partial establishment") the connection to this new destination as well.

- **Channel object**: There exists one Channel object for each channel; in the current prototype, the Channel object is co-located with its Target set object (i.e., these two objects are located on the same RCAP daemon), though a new proposal suggests that the Channel object should be located at the channel's source [64]. The Channel object maintains the following information about the channel: (a) source node, (b) traffic parameters, (c) the partition whose resources the channel can use, (d) for channels with resource sharing, the groups whose resources the channel can share in, (e) for advance-reserved channels, the channel start and end time, (f) current establishment state, and so on. The Channel object handles establishment and teardown, including incremental establishment/teardown when new destinations join (or existing destinations leave) the same channel, which is "live".

- **Group object**: There exists one Group object for each channel group; these group objects currently maintain resource sharing information: for each group, the corresponding Group object maintains the aggregate traffic specification and the sharing threshold as well as the list of member channels. These Group objects also maintain other information for optimizing routes to enhance resource sharing gains.

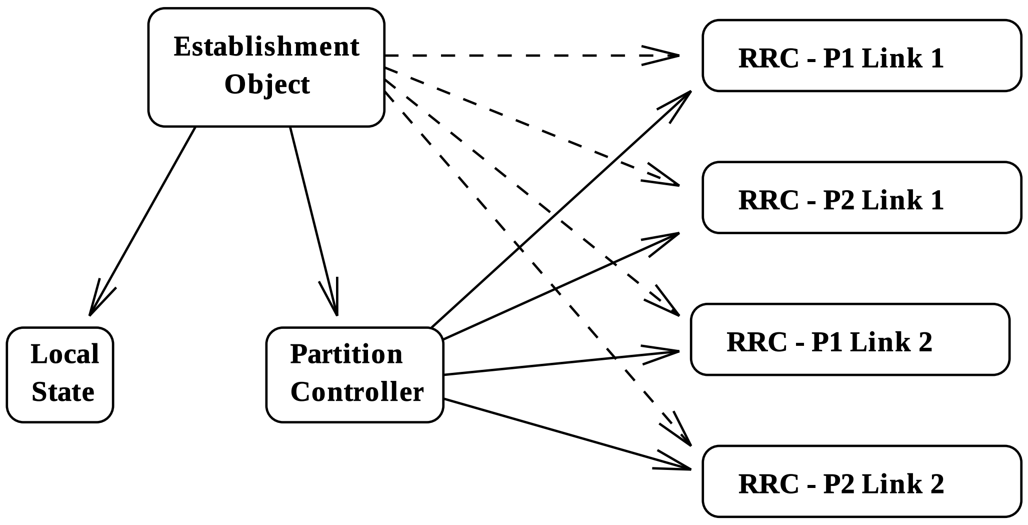

Figure 5.3: Establishment subsystem

Figure 5.3 shows RCAP daemon's establishment subsystem. When a Channel object receives a channel-establishment message, it sends a *Channel-establish* message to the Establishment object located at the channel source node. The establishment signaling and admission tests are performed by the establishment sub-system; this sub-system includes the following objects:

- **Establishment Controller**: There exists one Establishment controller per RCAP daemon. All establishment-related messages (to establish a new channel, to tear down an existing channel, or to make dynamic, incremental changes) go to the Establishment Controller (EC) on that node. The EC looks up the partition and routing information in the message to determine the servers that need to run the admission control tests, and contacts the Partition Controller to obtain pointers to the appropriate Resource Reservation Controllers (RRCs), each RRC performs admission control for one partition on one server. The EC obtains the results of admission control from the RRC, sets up the appropriate local state information in Local State objects (see below), and sends appropriate messages to the EC on the next node(s).

- Partition Controller : There exists one Partition Controller (PC) at each RCAP daemon; the PC maintains information about the various partitions on the local servers. It is responsible for storing creating new partitions, for controlling changes in allocations to the different partitions, and for ensuring consistency among these allocations during these changes. The PC knows about all the RRCs that exist at the node, and returns this information when EC requests it.

- Resource Reservation Controller : There exists one Resource Reservation Controller (RRC) per partition per server (CPU or outgoing link); for any partition on a server, the corresponding RRC maintains admission control information; this information includes:
  - list of resource sharing data for channels that belong to this partition-server
  - current resource allocations; this information is maintained as an interval table, with a pointer to the appropriate admission control server for each time interval.

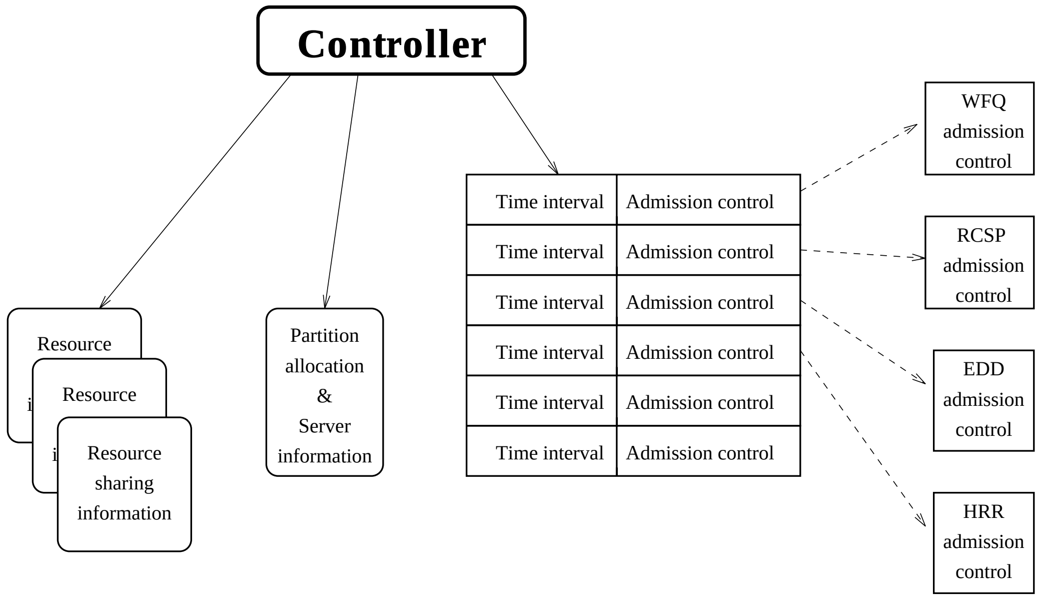

Figure 5.4: Resource reservation controller

Figure 5.4 shows the block diagram of an RRC; the RRC is responsible for maintaining information about resource sharing on a per-partition-per-server basis and for coordinating the admission control tests among the admission control server objects (see below). If the tests succeed, it also sets up tentative reservations in the interval table(s) and informs the EC whether the tests succeeded or failed, along with the tentative reservations (if the tests succeeded).

- Admission Control Server (one per interval table):

  The admission control server maintains state information for an interval table, and provides admission control methods for the same; these admission control methods depend on the scheduling discipline; in the same node, different admission control servers (ACS) may use different scheduling disciplines.

- Local State Object(s): In addition to the objects described above, the Establishment system includes local state objects that maintain information about the existing channels, including their traffic specifications, routes, and so on. This information is used for subsequent processing (changing the specifications, dynamically changing a channel's destinations, tearing down connection requests, and so on.)

### 5.1.5 Connection establishment in Tenet Suite 2

We now illustrate the sequence of RCAP activities for connection establishment with a simple example; we will use the example of Section 5.1.3.

At Step 1, the RCAP daemon receiving the `Create_Target_Set` request sends the request to the local Target Set Manager; this manager creates a new Target Set object and returns the object-id (`TS1`) to the client, along with the object's secret key. At Step 2, the RCAP daemon receiving the `Create_Channel` call forwards the request to the daemon where `TS1` is located; at this daemon, the request is forwarded to the Channel Manager which creates a new Channel object, records the relevant information (channel source node, partition, traffic specification and so on.) and returns the object-id (`ChA`) to the client (along with the channel's secret key); the `Create_Channel` calls of Step 3 are handled in a similar manner. The calls in Step 4 are forwarded to the appropriate Channel objects, which record the information provided (advance channel, start time, end time) within themselves.

At Step 5, the RCAP daemon receiving the `Create_Group` request sends the request to the local Group Manager; this manager creates a new Group object, records the group traffic specification and threshold, and returns the object-id (`G`) to the client (along with the group's secret key). The `Include` calls of Step 6 are forwarded to the appropriate Channel objects, which record the `Group Id` (`G`) within themselves.

The `Establish_Channel` requests of Step 7 are forwarded to the appropriate Channel objects. For channels with resource sharing, the Channel objects contact the Group object(s) (in our case, `G`) to get the group traffic and threshold specification; these channel objects also obtain routing-related information for maximizing route overlaps among themselves. The Channel object then contacts the routing system (via the Routing Stub object) to obtain a multicast route. It then composes a *Channel-establish* message; this message includes the channel traffic, partition, resource sharing related information (if any), and, for advance reserved channels, the start and end times. It then sends this *Channel-establish* message to the Establishment object at the channel source node.

At each node along the (multicast) route, the Establishment object parses the establishment message and requests the Establishment system to perform admission tests (we will describe that procedure a little later in this section), and, if successful, to make the reservations. If the tests fail, a *channel reject* message is sent back along the path; otherwise, the Establishment object updates and forwards the *Channel-establish* message to the Establishment objects at the next node(s) down along the multicast route. This process continues hop-by-hop to the various destinations; at each destination, the Establishment system performs admission tests to determine if end-to-end performance bounds can be met. Depending on these results, either a *channel accept* or a *channel reject* message is sent back to the previous node.

At each node along the return path, resources are committed for each branch along which at least one destination was reached, and released otherwise. When replies are received from all downstream branches, a *channel accept* message is sent back to the previous upstream node, except if all messages received were of the *channel reject* type, in which case a *channel reject* message is sent back. The Establishment object at the channel source then reports the results to the Channel object, which then returns this information to the client.

We now describe, in detail, the sequence of actions in the Establishment system for admission control during connection establishment.

1. From the *Channel-establishment* message, the EC determines the *Partition* that the channel belongs to, along with the routing information, from which it determines the servers on which the admission tests should be run.

2. The EC contacts the Partition Controller (PC) with the $< PartitionId, ServerId >$ tuple(s), and obtains the pointers to the appropriate RRCs.

3. The EC invokes the RRC, passing the channel parameters (e.g., traffic, reservation time parameters, resource sharing).

4. The RRC determines whether resource sharing can be used to bypass running admission tests; if not, the RRC determines the intervals over which the tests should be run. This process may also split up to two interval tables.

5. The RRC invokes admission control servers for all appropriate time intervals, and invokes a *resolver* on the returned results, which include information about whether the tests succeeded, and, if the tests succeeded, the local performance parameters, including bounds on delay and jitter; the resolver selects the *bounding value* for the results.

6. If all tests succeed, the RRC sets up tentative reservations in the interval table(s).

7. The RRC returns the admission control results to the EC, which, if the admission control tests succeed, sets up local state information and sends an updated *channel-establishment* message to the next node along the channel route. If the admission control tests fail, the EC sends back an *establishment-fail* message.

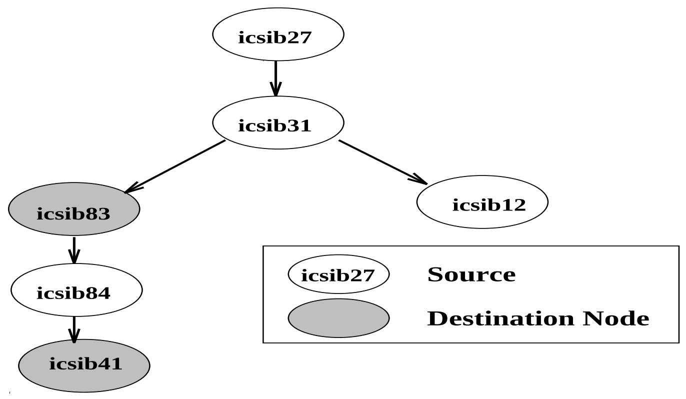

Figure 5.5: Network topology for the experiment

8. When the *channel-establishment* message reaches a node which is also a destination for the channel, the EC invokes *destination tests* to determine whether the end-to-end performance requirements were met, and sends back *establishment-success* or *establishment-fail* message accordingly.

9. When a node receives all *establishment-success/fail* messages from the downstream nodes, it sends back an *establishment-success/fail* messages back to the upstream node; an *establishment-fail* message is sent back only if establishment failed for all downstream nodes.

### 5.1.6 Suite 2 RCAP measurements

One concern often expressed about admission control is that the admission tests might add considerably to the cost of connection establishment. To address this issue, we measured round-trip delay, per-hop latency, and the time spent in running admission tests on a prototype implementation; this implementation (over 40K lines of C++ code) includes the resource sharing, resource partitioning, and advance reservation mechanisms that we described in the previous three chapters. We now report the results obtained with a network of Sun `Sparc 1` class workstations. The measurements were collected using the logical topology shown in Figure 5.5, which contains one branch node and three destination nodes; the longest path contains four hops. The measurements were collected using a simple tree topology with one branch node and three destination nodes; the maximum number of hops (from source to any destination) is four. Table 5.1 shows the typical ranges of measured establishment delays; in this Table, the start-up latency includes the time from the Channel object receiving the Establishment request to the time the Channel-establish messages is sent to the Establishment object.

For this tree topology (over ethernet) the total round-trip delay for channel establishment is 110 to 130 ms; the channel establishment time is dominated by message transport delays and the user-kernel overhead. We broke down the channel establishment

| | Round-trip delay | Start-up latency | Per-node latency | Communication overhead (TCP/IP) |
|---|---|---|---|---|
| Non-advance | 110 - 125 | 8 - 10 | Forward : 5 - 7<br>Reverse : 1.5 - 2.5 | 50 - 60 |
| Advance | 115 - 130 | 8 - 10 | Forward : 5 - 7<br>Reverse : 1.5 - 2.5 | 50 - 60 |

Table 5.1: Channel establishment times (ms)

latency into per-node components; as mentioned in Table 5.1, setting up the state information during the forward pass takes about 6ms per node, and the confirmation on the reverse pass takes about 2ms per node. The admission tests take less than 1ms per node, which is small compared to the channel establishment latency[3].

These results are similar to the connection establishment latency for Tenet Suite 1 without advance reservations, where the establishment time was about 80 to 90 ms for unicast channels with six hops[4] [7].

These measurements were made on our un-optimized initial prototype; the excessive communication overhead (50-60ms) is due to the manner in which we implemented inter-RCAP communication in this prototype, which is done via reliable TCP messages. In our prototype implementation, for simplicity and ease of implementation, the RCAP daemon opens a new TCP connection whenever it wants to send a message to another RCAP daemon, and closes this TCP connection after sending the message. Thus, every RCAP message transmission picks up this extra TCP connection open/close overhead. We believe that we can reduce the connection establishment time by 30-35ms by keeping open the TCP connections used for signaling (for example, to adjoining neighbors or by caching these connections).

## 5.2 RMTP and RTIP

RTIP is the network layer data delivery protocol in the Tenet protocols; RTIP operates at each host, switch and gateway along the channel's route, and performs rate control, jitter control, packet scheduling, and data transfer functions [118, 126, 129]. As described before, RTIP provides simplex, sequenced, unreliable, performance-guaranteed packet delivery service. A packet may not be delivered correctly to a connected destination for two reasons: (a) the packet may be corrupted during transmission, or (b) due to buffer overflow (for example, if the channel traffic exceeds the pre-negotiated traffic bounds). As the client data is not checksummed, the packets may get corrupted in transmission ("bit-errors on the wire").

[3]We obtained similar results with our measurements on a network of DEC 5000 (20-40MIPS) workstations.

[4]The Suite 1 measurements were over a wide-area-network, where the propagation delays contributed about 30ms to the channel establishment latency.

RMTP is the transport layer data delivery protocol in the Tenet protocols; RMTP operates at the end-nodes (senders and destinations) and uses RTIP to provide a simplex, end-to-end, unreliable, in-order, performance guaranteed message delivery service. RMTP tasks include message fragmentation and re-assembly, and traffic policing. Figure 5.6 shows the software structure for RMTP/RTIP. As RMTP does not require any changes for supporting multi-party communication, we will not discuss it any further in this chapter.

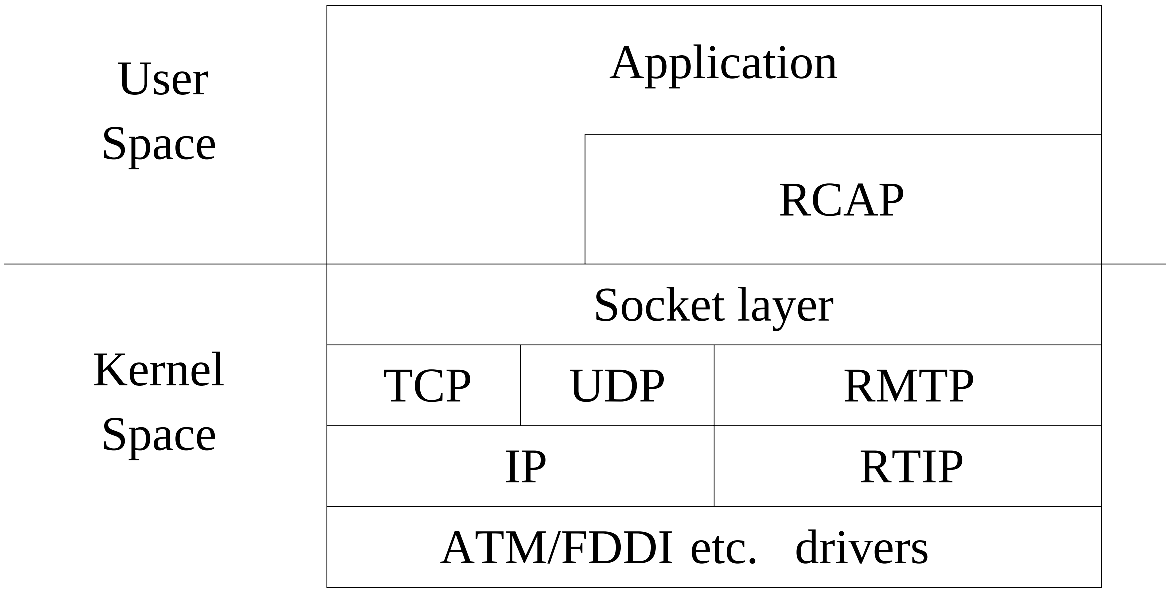

Figure 5.6: Software structure of RMTP/RTIP (courtesy Hui Zhang)

To illustrate the changes in RTIP for supporting multi-party communication, we first present a simplified view of the unicast Suite 1 RTIP. Real-time channels must be established before they can be used to transmit data. For setting up the channel at a particular node, the RCAP daemon must inform the RTIP at that node[5]. This invocation sets up the mapping from incoming Virtual Channel Identifier (VCI) to the outgoing link and VCI in the RTIP forwarding table. During this invocation, the RCAP daemon passes the following information to the RTIP entity: the incoming VCI, the outgoing VCI, the outgoing link-id, the channel traffic specification (for traffic policing), and the local performance bounds.

When RTIP receives a packet, it performs the following functions:

1. Compute packet header checksum; verify header consistency.

2. Use the incoming VCI to look up current state (for traffic policing) and the forwarding table to determine the next node[6] and VCI.

[5]The RCAP daemon is a user-level process; the RTIP entity is in the kernel. This invocation is accomplished via a *setsockopt* call.

[6]Please note that this is RTIP for unicast channels.

3. Check the clock and compute the packet deadline[7] and holding time; update traffic policing state.
4. Update the headers (next node VCI); recompute header checksum
5. Schedule packet for transmission in the priority queue.

### 5.2.1 RTIP changes for multi-party communication

RTIP requires two changes to support multi-party communication: (a) multicasting requires that RTIP send copies of incoming packets on multiple outgoing links at branch nodes, and (b) supporting resource sharing requires changes in the traffic policing and data forwarding mechanisms[8]. We now discuss each of these changes in turn.

**Multicasting**: Multicasting requires RTIP to forward incoming packets on multiple outgoing links at the branch nodes. To support this multiplicity in forwarding information in the Suite 2 RTIP, the forwarding table is modified to provide a linked list, where each element provides an outgoing link and VCI. When a packet is received, in principle, RTIP looks up the forwarding table, determines the number of outgoing links, makes that many copies of the incoming packet, and schedules each copy for transmission on the appropriate outgoing link. In practice, RTIP code is designed to avoid making unnecessary multiple copies.

**Resource sharing**: As described in Section 2.2, to support resource sharing, RTIP must support traffic policing and rate control on the aggregate group traffic specification. Before resource sharing, RTIP could set up the traffic policing and rate control state information on a per-channel basis (and index it by VCI); with resource sharing, RTIP must set up this state information on a per-group basis for the channels with group resource reservation, and also allow these different channels to share this state information.

Supporting this shared "group state" information requires an additional indirection in the RTIP code. Instead of the forwarding table including (and providing direct access to) the policing-related state information, we now have a separate "state table" which maintains the policing state information, and the forwarding table includes pointers to this state table. This state table has one entry for each group of channels, and one entry for each channel that is currently not in any resource sharing relationship. The same entry in this state table can be pointed to by many different channels in the forwarding table; in this way, the state information can be shared.

### 5.2.2 RCAP-RTIP interface

As we described in the previous sub-section, the RCAP-RTIP interface in Suite 1 is via *setsockopt* calls; we now describe how these calls change for multi-party communication. In Suite 1, the *setsockopt* call for channel setup is

[7]Assuming that deadline scheduling is used.

[8]As we mentioned before, RTIP does not require any changes for supporting advance reservations and/or resource partitioning; these mechanisms only require changes in RCAP.

```
setsockopt(sock, IPPROTO_RTIP, RTIP_SPEC, rtip_spec, length)
```

where the rtip_spec data structure includes: (a) the incoming VCI, (b) the outgoing VCI, (c) the outgoing link-id, (d) the channel traffic specification (for traffic policing), and (e) the local performance bounds.

For channel teardown, the *setsockopt* call is

```
setsockopt(sock, IPPROTO_RTIP, RTIP_RELEASE, vci, length)
```

where *vci* provides the incoming VCI for the channel at that node.

As we mentioned in the previous sub-section, there are two changes in RTIP: for multicasting and for resource sharing; the new RCAP-RTIP interface must provide for these changes.

For multicasting, we can set up an initial forwarding table entry, and later add more entries for the same channel (adding new branches to the multicast tree when a new destination joins). Similarly, an existing destination can leave a multicast distribution tree; in this case, RCAP needs to remove one branch of the tree without disrupting the data flow to the other destinations. To support resource sharing, the RCAP-RTIP interface must allow RCAP daemon to pass sharing-related information to the RTIP entity in the kernel; for a resource sharing channel, this includes the group-id for the group whose "traffic policing state information" that channel should use, along with the group aggregate traffic specification. Also, this interface is required to support dynamic changes in the channel traffic specification [94, 93, 92, 91]; this implies that RCAP be allowed to change the traffic specification of "live" channels[9].

In designing the new interface, a key concern was to minimize the potential error states as well as the interface complexity. Another key consideration was compatibility with the existing RCAP-RTIP interface; in principle, Suite 1 RCAP should be able to talk to Suite 2 RTIP, with minimal changes (or through a simple "filter").

For channel setup, Suite 2 adds one parameter to the rtip_spec data structure in the setup (RTIP_SPEC) setsockopt call: AllocId. A channel (at a node) is uniquely identified by its AllocId; in addition, at every node at which a resource sharing group is used, that resource sharing group is uniquely identified by its AllocId, which thus relates (for resource sharing) different channels. When there is resource sharing, all channels (that belong to that group) use the same AllocId (the group's AllocId). The traffic specification corresponds to the AllocId specified in that call.

The teardown (RTIP_RELEASE) call adds one new parameter: the outgoing link-id. When RTIP gets the RTIP_RELEASE call, it releases the resources associated with the channel at that outgoing link. However, the state information associated with the channel is maintained till all outgoing links go away. Similarly, each "traffic policing state" entry keeps track of the number of channels sharing that entry; when the number goes down to zero, the state entry is deleted.

[9]This dynamic change will also take place when a channel switches between using channel and group traffic specification.

### 5.2.3 An example

We now describe a simple example to illustrate the RCAP-RTIP interface, along with the sequence of actions in RTIP for setting up the forwarding tables. We assume requests for three channels in a node, each with traffic requirement 1.5 Mbps[10] where these channels can share resources, with overall group requirement 4.0 Mbps; this is similar to the example presented in Table 3.1 in Chapter 2. We also assume that the sequence of channel setup requests at this RCAP daemon is:

1. Channel A, to outgoing link L1
2. Channel A, to outgoing link L2
3. Channel B, to outgoing link L1
4. Channel C, to outgoing link L1

The last request (for Channel C) will trigger resource sharing for outgoing link L1.

Table 5.2 shows the sequence of setup (RTIP_SPEC) calls with the corresponding parameter values.

Table 5.2: Sequence of setup (RTIP_SPEC) call parameter values

| Due to call | For channel | Link | Traffic spec. (Mbps) | AllocationID |
|---|---|---|---|---|
| 1 | A | L1 | 1.5 | A |
| 2 | A | L2 | 1.5 | A |
| 3 | B | L1 | 1.5 | B |
| 4 | **C** | **L1** | **4.0** | **G** |
| 4 | **A** | **L1** | **4.0** | **G** |
| 4 | **B** | **L1** | **4.0** | **G** |

The sequence of events in RTIP is as follows:

1. When RTIP receives the first call, since it has never seen Channel A before, it sets up a new forwarding table entry; it also adds a forwarding entry for link L1 in this table. Since RTIP has not seen AllocationID A for link L1 before, it sets up a new State table entry for this channel, and sets up the pointer from the forwarding table to this entry.
2. When RTIP receives the second call, since it has seen Channel A before, it re-uses the existing forwarding table entry; it adds a forwarding entry for link L2 in this table. Since RTIP has not seen AllocationID A for link L2 before, it sets up a new State table entry for this channel, and sets up the pointer from the forwarding table to this entry.

[10]For ease of explanation, we are using a simplified traffic specification that only uses peak-rate bandwidth; the actual traffic specification is more complex.

3. When RTIP receives the third call, since it has never seen Channel B before, it sets up a new forwarding table entry; it also adds a forwarding entry for link L1 in this table. Since RTIP has not seen AllocationID B for link L1 before, it sets up a new State table entry for this channel, and sets up the pointer from the forwarding table to this entry.

4. When RTIP receives the fourth call, since it has never seen Channel C before, it sets up a new forwarding table entry; it also adds a forwarding entry for link L1 in this table. Since RTIP has not seen AllocationID G[11] for link L1 before, it sets up a new State table entry for this channel, and sets up the pointer from the forwarding table to this entry.

5. When RTIP receives the fifth call, since it has seen Channel A before, it re-uses the existing forwarding table entry; furthermore, since the forwarding entry for link L1 exists, it changes that entry's state table pointer to point to the State table entry for AllocationID G.

6. When RTIP receives the sixth call, since it has seen Channel B before, it re-uses the existing forwarding table entry; furthermore, since the forwarding entry for link L1 exists, it changes that entry's state table pointer to point to the State table entry for AllocationID G.

## 5.3 Summary

In the previous three chapters, we described resource sharing, resource partitioning, and advance reservations; these constitute the core of our multi-party real-time communication system. In this chapter, we described, with the help of several examples, how we integrated these mechanisms in the Tenet Protocol Suite 2 to provide network support for multi-party real-time communication.

Supporting resource sharing required significant changes in the client interface; we separated channel creation from channel establishment, and this separation also simplified the client interface for specifying advance reservation of network resources; this interface also mapped well to our RCAP daemon organization. The resource partitioning and advance reservation-related information is maintained at the Channel object while the resource sharing-related information is maintained at the Group object; this sharing-related information is collected by the Channel object at connection establishment time, and shipped in the *Channel establishment* message. In keeping with our design principles, all resource sharing-related decisions are made in a fully-distributed manner; the partition and advance reservation-related decisions are also made in a fully-distributed manner. For example, if a partition does not have the resources to support a channel, the RCAP daemon may decide to borrow resources from another partition for supporting a new channel request[12]. This

[11]AllocationID G is for the resource sharing group.

[12]This borrowing is subject to network administration and management policies, as described in [45, 58].

design also promotes high flexibility and independence in decision-making, as different outgoing links on the same node may make different decisions. We then showed how the client application requests are mapped to the various activities in the network communication system for supporting multi-party real-time communication.

# Chapter 6

# Related work

In the previous chapters, we discussed the three key components of our multi-party real-time communication research: *resource sharing*, *resource partitioning*, and *advance reservations*, and how these components work together in Tenet Suite 2 to provide efficient and useful network services for for multi-party real-time applications. Resource sharing mechanisms provide efficient support for large-scale conferences; resource partitioning provides an effective tool for the network managers to control and distribute resource allocation among the various network service clients; advance reservations enhance the service usability, and at the same time, can help improve the network performance with better planning and off-line routing potential.

Network support for multi-party real-time communication has been a very actively researched area for the last several years; several other researchers have designed techniques and mechanisms that take alternative approaches and trade-offs for supporting these applications. Though it is instructive to see how the different design goals and considerations led to the differences in the mechanisms designed, a comprehensive discussion of all the issues and approaches and mechanisms in this area would be too long. For this chapter, we therefore decided to restrict ourselves to the work directly related to the three components of this research: resource sharing, resource partitioning, and advance reservations.

Analogs to resource sharing exist in the filters of Resource reSerVation Protocol (RSVP) and in the bandwidth sharing stream groups of the Stream Transport 2+ (ST2+). Similarly, Class-Based-Queueing (CBQ) may be viewed as being somewhat similar to resource partitioning. Advance reservation mechanisms have been proposed for ST2 [102, 37] as well as for RSVP [30]. However, the overall framework and design goals significantly impact the design choices; also, it is important to see how these mechanisms work together to provide a useful and efficient service to the multi-party applications. Due to these reasons, we have organized this chapter on a per-scheme basis. We will first compare and contrast the ST2 design with ours. We will then conclude this chapter with a discussion of Integrated Services Internet Protocols (ISIP), under which the RSVP and CBQ protocols are being designed by the Internet research community.

## 6.1 ST2

IP multicasting provides support for multi-party communication but it does not provide any kind of QoS assurances. ST (Stream Transport Protocol) was the first attempt to provide some kind of QoS assurances in the multi-party communication environment. The specification of the first version of ST was published in the late 1970's and the protocol was used during the 1980's for experimental voice and video transmission. The experience gained in those experiments led to ST-II, the revised Stream Transport protocol. This work was done primarily by the Internet Engineering Task Force, and a comprehensive description of ST-II can be found in [114]. The ST-II standard has been substantially modified in the new release; the protocol is now dubbed ST2+[31]. We will use the name ST2 to refer to the features and the mechanisms that are common to both these protocols.

ST2 permits a sender to establish a multicast connection (tree) to one or more receiving targets. Nodes in the tree represent ST2 agents which execute the ST2 protocol. The sender can send a continuous stream of packets that are appropriately forwarded by the ST2 agents following the pre-defined routes.

ST2 was designed to inter-operate with a wide variety of networks. Consequently, it made few assumptions about the services provided by the underlying data-link and physical layers, and when any assumptions were made, the designers attempted to define correct behavior in the event that the underlying network did not provide those features. Thus, while ST2 can use any QoS assurances provided by the underlying layers, it also puts in a lot of effort to ensure that the protocol can run (without offering any guarantees, of course) over networks that cannot provide such guarantees.

In the original ST2 transport model, the source specifies the QoS requirements for the stream at connection establishment time. Each ST2 agent that receives the specification makes appropriate reservations, after possibly modifying the QoS request if it cannot provide the requested QoS. These messages flow downstream, to the destinations, which decide whether the final QoS assurances are acceptable. If a target accepts the connection, the final QoS specification is propagated back to the source. In the newer versions (modified ST2 as well as ST2+), if the senders sets the permissions appropriately, the receivers can also specify their QoS requirements, and join data streams without an explicit sender notification [31, 34, 36].

The initial ST-II protocol did not have any provision for resource sharing even though it specified *Stream groups* which were equivalent to the *channel groups* that the Tenet protocols use to specify resource sharing. ST2+ permits the applications to define *bandwidth sharing* stream group(s); these groups can be used for specifying resource sharing relationships. In the ST2+ protocol, the resource sharing specification is very restricted. The application merely specifies $N_{max}$, the *maximum concurrency*; the network then computes the aggregate traffic specification by multiplying $N_{max}$ with the bandwidth required by the most demanding application to compute the aggregate bandwidth required. This can also lead to a potential problem in that specifying such bandwidth sharing may actually result in a request being denied when, without such sharing specification, the same request would have been accepted. Also, the resource sharing system is not integrated with the rest of the network; therefore, the routing system cannot attempt to maximize the resource sharing gains.

Several researchers have investigated providing advance reservations under ST2. [102] proposes a signaling protocol for this purpose; the mechanism is similar to our interval-table based advance reservation system, except that it relies on fixed-size intervals. The fixed size intervals can lead potentially to very high computational and state management overhead for accessing and maintaining the interval tables; we described this problem in Chapter 4. [37] provides a good description of the various issues in reserving resources in advance, in the context of ST2. However, we are not aware of any implementations (or even simulations) of advance reservation mechanisms done by the authors of that paper.

The heavy emphasis on compatibility was ST2's strong point; it was also ST2's undoing. The lack of good multi-party applications and hardware platforms severely hindered the ST2 designers. ST2 also permitted "subset" implementations to allow for easy implementation and experimentation. Unfortunately, this led to more than fifteen different implementations of ST2 which were mutually incompatible! This situation was rectified in ST2+ which mandated that all "correct" implementations must support the *full* protocol.

## 6.2 Integrated Services Internet Protocols

A different approach has been taken by the Internet researchers; the protocol suite is called Integrated Services Internet Protocols (ISIP) and the project includes researchers from MIT, Xerox PARC BBN, USC/ISI and Stanford University. In this approach, the system architecture is divided into five components:

1. flow specification, which describes both the characteristics of the traffic stream sent by the source, and the service requirements of the application; the proposed *flow spec* is described in [96];

2. routing, which refers to the mechanisms that provide good unicast and multicast paths for data flow; the routing protocols under consideration include Protocol-Independent Multicast (PIM) [21, 24, 25, 23, 22] and Distance Vector Multicast Routing Protocol (DVMRP) [1, 17, 39, 26, 120, 122];

3. resource reservation protocol, which refers to the protocol by which the resources are reserved inside the network for providing the QoS assurances; this ongoing research has led to the design of RSVP [130, 108];

4. admission control, which refers to the algorithm that determines which reservation requests to grant and which to deny, thereby maintaining the network load at an appropriate level [69]; and

5. packet scheduling, which refers to the algorithm for selecting the next packet that is serviced at a particular node, and when that packet may be served; two such scheduling disciplines are FIFO+ [18] and WFQ [88].

The overall framework for this research is the Integrated Services (int-serv) Working Group of the Internet Engineering Task Force (IETF) [9]. The int-serv group's goal is to provide support for guaranteed as well as "predictive" and other services in a consistent, integrated manner in an inter-network. A key component of this approach is the use of CBQ to logically support multiple virtual subnetworks with different sets of requirements [121]. The work is currently in progress and it is yet not clear how well these different modules will work together; we will limit this discussion to CBQ and RSVP because they are the only relevant ones to the subject of this dissertation.

### 6.2.1 RSVP

RSVP takes a very different approach to "flow"[1] setup and management. Instead of requiring the senders to manage the flow, RSVP has each receiver manage its part of the flow's distribution tree. This receiver-oriented design was prompted by the following seven design goals at the initial stages in the RSVP design process [130]:

1. Accommodate heterogeneous receivers.
2. Adapt to changing multicast group membership.
3. Exploit the resource needs of different applications to use network resources efficiently.
4. Allow receivers to switch channels.
5. Adapt to changes in the underlying unicast and multicast routes.
6. Control protocol overhead so that it does not grow linearly (or worse) with the number of participants.
7. Make the design modular to accommodate heterogeneous underlying technologies.

RSVP introduced several interesting features: receiver-oriented reservation as well as maintaining "soft state" in the network[2]. Soft-state is information that is periodically refreshed by the interested parties; in this case, the senders and the receivers are required to periodically update the state at the routers. This allows the protocol to recover gracefully from system failures in the network; it also allows the protocol to dynamically adapt to changes in the network routing system. The problem, though, is that the reservations are no longer guaranteed. When the routes change, there are no reservations along the new paths till the state is refreshed (typically about 30 ms); also, there is no guarantee that resources will be available along the new path.

RSVP is being designed as a component of the ISIP protocols; a key design goal is that it be able to work with other existing (and future) technologies used in the Internet and other networks. To achieve this independence from the routing system, RSVP merely queries the routing system and uses the routes that the routing system provides. The routing system is free to change these routes at any time; for each flow, RSVP periodically queries

[1]the flows in RSVP are somewhat similar to the streams of ST2 and the channels of the Tenet protocols.

[2]Soft-state was first described in [16]; RSVP was the first protocol to use soft state.

the routing system to dynamically adapts to these route changes. RSVP also depends on the IP multicasting to correctly forward all packets; it merely sets up the reservations that provide better service to these packets. Thus, the conference participants must first use IP multicasting to set up a "best-effort" conference and then invoke RSVP to set up (and periodically refresh) the reservations along these paths.

The receiver-oriented RSVP requires that the resource sharing relationships be specified by the receivers; for specifying these relationships, RSVP supports at least three distinct "filtering styles". With *Fixed filters*, the receivers specify a list of senders, along with the *distinct* reservation for each sender. With *Shared Explicit*, the receivers specify a list of senders, along with the *shared* reservation for the group of senders. With *Wildcard filters*, the receivers specify a multicast group (for example, the IP HostGroup), with the *shared* reservation. All senders sending data to that multicast group automatically become eligible for sending data using that filter. The initial RSVP specification also supported *Dynamic filters*, but the dynamic filtering style has been removed[3] due to the problems in implementing the software for supporting it.

RSVP receivers send these reservation messages (RESV) upstream to the senders the senders for that "session". The RESV messages belonging to the same IP multicast groups are merged together[4]as they travel upstream; this merging is essential for reducing the network traffic overhead to manageable levels. By using these receiver-specified filters and by merging these filters as the RESV messages move upstream, RSVP avoids the problem of managing and accessing the *Sharing Group* object of the Tenet Suite 2. This avoidance becomes especially useful as RSVP does not have to worry about handling failures that may make this Group object unavailable; all information is available at the receivers and this design is thus in keeping with the *fate-sharing* principle [16].

On the other hand, this avoidance has associated costs. It is quite possible that different receivers may specify different reservation levels for the same set of senders. Since many data streams cannot be scaled arbitrarily without serious degradation in perceived quality (e.g. [87] ), we feel that the source should specify the resource requirements (at least in terms of bandwidth). Receiver control over bandwidth requirements can be obtained by using layered coding schemes [80, 101, 54, 12, 119] and putting each layer in a separate sharing group. Also, if different receivers specify different sets of senders in their specifications, the merging can cause the system to violate the performance guarantees provided to the users. Finally, merging *Wildcard filters* can cause some serious looping problems [6]. The current RSVP proposal handles this problem by introducing "SCOPE" objects which list the set of senders in the RESV messages; this can lead to significant scalability problems.

[30] proposed an advance-reservation service for predictive service under RSVP. Since this work did not include any resource partitioning mechanisms, they ran into serious starvation problems (advance reservations starving out immediate requests). This advance reservation proposal has not been accepted by the RSVP Working Group at the IETF.

[3]Downgraded to "for further study" in the RSVP Working Group meeting [10]

[4]Different filtering styles are incompatible; hence, RESV messages with different filtering styles are not merged together.

### 6.2.2 Class Based Queueing (CBQ)

Sally Floyd and Van Jacobson have devised Class-Based-Queueing (CBQ) as a "link-sharing" scheme [50, 121, 51]. The key design goal for their effort was to provide a network with the ability to support a reasonable distribution of network resources between the different entities paying for the network. Connections belonging to each paying entity get classified into a separate class. With their scheduling discipline, they provide a per-packet data traffic monitoring and control facility for ensuring that during congestion, the different paying classes (or customers) can obtain their expected share of the link bandwidth. When a particular class does not require its share of link bandwidth, the same is made available to the connections belonging to other classes. The basic link-sharing idea has been extended to include support for hierarchies of classes.

For guaranteed-performance communication, the CBQ scheme is fairly similar to our resource partitioning scheme; the main difference is that we designed our scheme for connection-oriented real-time protocols, while CBQ is designed to work with connectionless protocols. This is why CBQ needs a packet classifier at each node during data delivery; we can use the connection identifiers that are already available to us. Due to this difference, CBQ interacts with packet scheduling (for packet classification) while our resource partitioning scheme works without affecting data delivery at all. Also, because of the distinction between channel identifiers and partition identifiers, we can change a channel's partition dynamically without affecting the packet delivery in any manner; CBQ will require adding an extra level of indirection to permit the same. Our routing system interacts with the resource partitioning system to generate feasible routes; to the best of our knowledge, these interactions have not been studied for CBQ.

## 6.3 Other efforts

Pasquale et al. have proposed a stream *filter* that is "... an executable module which may be placed on a port, and implements a function which takes a specific set of streams associated with that port and produces a new stream" [98]. These filters perform an *application-level* transformation of one or more streams. A multiplexing filter could perform a function similar to resource sharing by taking in streams from upstream nodes and multiplexing them on to a single output stream. It should be noted, however, that our resource sharing design does not require application-level modules within the network. Also, we are not aware of any implementation or any further research on these filters.

Some researchers believe that guarantees are too expensive to provide and that most people will be willing to tolerate occasional degradation in the QoS provided if that results in a significantly cheaper network service [18, 105, 95]; they have consequently devised other techniques and mechanisms to support these applications without providing performance guarantees [70, 69]. It is not yet clear if these mechanisms will suffice, or if these mechanisms will provide *cheaper* service than that with performance guarantees, or if these *adaptive* mechanisms will be stable.

Although Asynchronous Transfer Mode (ATM) [11, 15] is expected to be the dominant technology in the near future for supporting real-time applications, the ATM standards are currently under development; the current standards do not support multi-party applications and consequently the resource sharing and advance reservation issues have not arisen yet. Our concept of partition is similar to that of virtual path (VP) in ATM networks [104, 68, 103, 52], in that both a VP and a partition can be regarded as a set of connections. However, the connections in all partitions are real-time connections, whereas the virtual circuits in a VP do not have to be real-time Virtual Channels; our approach applies only to the real-time part of the traffic. Also, while a VP is defined over a specific route, which is the route of all its VCs, a partition can be defined over any fraction of the network (or internetwork), including the whole network; thus, a set of VPs, such as those forming a virtual private network (VPN), can be defined as a single partition. Another difference between partitions and VPs is that VP identifiers are used by switches to route cells, while partitions do not influence switching, routing, or forwarding, as they are a purely setup-time concept. Finally, while partitions can be nested (i.e., a partition can be subdivided into sub-partitions, and so on), the VP-VC hierarchy has been designed to have only two levels.

# Chapter 7

# Summary and suggestions for future work

## 7.1 Dissertation summary

In this research, we study the issues and tradeoffs that impact the design of network services to support multi-party real-time communication in integrated-services packet-switching networks; this dissertation presents some novel mechanisms for supporting multi-party real-time communication in packet-switching networks.

In Chapter 1, we presented the prior work done by the Tenet Group at Berkeley in designing and building network protocols for supporting unicast real-time communication. These protocols serve as the framework for our research; with this framework, we described a few salient issues that arise in multi-party communication: managing multicast group membership, supporting dynamic changes in these groups, and supporting heterogeneity in receivers. We then introduced the ideas that form the basis of this research effort: exploiting the characteristics of multi-party communication to improve the efficiency in using network resources; providing the network managers with effective ability to control the network resources; and providing a more usable service to the network users.

The traditional approach to supporting real-time communication allocates network resources to individual connections; this approach provides well-defined performance guarantees that are independent of other network traffic. To improve network resource utilization without sacrificing well-defined guarantees, Chapter 2 described *resource sharing*; resource sharing is based on the simple observation that in multi-party conferences, the participants usually co-operate. This co-operation is then exploited to reduce the network resource allocation for such multi-party applications; this increased resource allocation efficiency is critical for supporting large conferences. We described the three components of the resource sharing mechanisms: the client-service interface for the applications to explicitly specify potential resource sharing, the changes in the resource reservation and admission control system to support resource sharing (during the connection set-up phase), and the changes in the data delivery protocol to ensure that the applications' real-time guarantees are met even when other applications use resource sharing.

We evaluated the performance gains due to resource sharing by analysis as well as by simulation. The analysis provided useful lower bounds on performance gains in many different cases, including sparse networks as well as dense networks with bottlenecks and hot-spots. The analysis also showed that the routing protocols can significantly impact the resulting resource sharing gains; dynamic, load-balancing routing algorithms can reduce the resource sharing gains, while a "sharing-aware" routing system can significantly improve the resource sharing gains. The simulation results quantified the resource sharing gains and confirmed the analytical results: resource sharing is very useful in reducing network resource allocation.

Resource sharing is a critical and integral component of our multi-party real-time communication system; it is critical that resource sharing mechanisms work well with the other components of the full system; we concluded Chapter 2 by discussing some of these interactions.

Chapter 3 presented resource partitioning. For operational networks, it is important that the network service managers be able to control the distribution and apportionment of network resources among groups of users and/or classes of application; our resource partitioning mechanisms provide such capability to the network service providers. These partitioning techniques work at the connection set-up and resource reservation stage; they do not affect the data delivery protocols at all. Since the resource reservation and admission test algorithms at a network server depend on the packet scheduling discipline followed there, the corresponding *partitioned* admission tests also depend on the packet scheduling discipline. In this chapter, we provided, with proofs, the admission tests with resource partitioning, for a spectrum of packet scheduling disciplines, including Earliest-Due-Date (EDD) [49], First-In-First-Out (FIFO) [125], Rate-Controlled-Static-Priority (RCSP) [125], and Weighted-Fair-Queueing (WFQ) [88].

We evaluated the performance of our resource partitioning mechanisms via simulation; the simulations show that our techniques are useful and efficient, and the mechanisms work well. The *resource allocation fragmentation losses*[1] are reasonably small, and these resource partitioning mechanisms can substantially reduce the computational overhead associated with running admission tests. Again, it is critical that resource partitioning mechanisms work well with the other components of the multi-party real-time communication service; we concluded Chapter 3 with a discussion of these interactions.

The ability to reserve real-time connections in advance is essential in distributed multi-party applications using a network which controls admissions to provide good quality of service. Chapter 4 presented the design and evaluation of our mechanisms for supporting advance reservations. We first discussed the requirements of the clients of an advance reservation service, and a distributed design for such a service. It is interesting that in addition to providing a much-needed service to these applications, the advance reservation mechanisms also improved the network service with better planning (network dimensioning) and routing [47].

We evaluated the performance of our advance reservation mechanisms via simulations; the simulation results demonstrated the usefulness of the mechanisms that we

[1]These losses are defined and described in Chapter 3.

designed. These simulations also provided useful data about the performance and some of the properties of these mechanisms. Again, it is critical that advance reservation mechanisms work well with the other components of the multi-party real-time communication service; we concluded Chapter 4 with a discussion of these interactions.

In Chapters 2, 3, and 4, we present resource sharing, resource partitioning, and advance reservations, the three cornerstones of our multi-party real-time communication research. In Chapter 5, we described how these components fit together to provide multi-party real-time communication service in the Tenet Scheme 2 (and the associated protocols, the Tenet Suite 2). We first discussed the design goals for the signaling protocol (RCAP) and described how these design goals led to our design decisions. We then described the object-oriented design of RCAP software and illustrated these interactions with a simple connection establishment example, along with some preliminary measurements on our prototype implementation.

Supporting multi-party applications also requires two key changes in RTIP, the real-time data delivery protocol: for multicasting and for resource sharing. We described these changes also in Chapter 5, and we concluded that chapter with a discussion of some of the interactions between resource partitioning and advance reservations in our implementation.

Chapter 6 reviewed related work by other researchers; we contrast our approach and research with that done by the designers of RSVP and ST2+. We first described the differences in the design goals for each project, and we then described how these different objectives led to the differences in the selected mechanisms.

## 7.2 Suggestions for future work

Although we have made important progress in understanding issues and tradeoffs in the design of network protocols for supporting multi-party real-time communication, a number of issues still need to be explored further.

We have shown that resource sharing can significantly improve network resource utilization. However, a few important concerns remain. Firstly, when the RCAP daemon (or, the Local Resource Manager of ST2+) decides to "switch" from using individual resource allocations to group resource allocations (and vice versa), the packets in transit may experience extra jitter and/or re-ordering, especially if this transition requires changing some of the local performance parameters. This phenomenon also depends on the packet scheduling policy followed; with EDD, it is relatively simple to use extra buffers to address this problem. It will be interesting to analyze other scheduling disciplines to design mechanisms and/or policies to address this issue.

Recently, some researchers have argued that for efficiently supporting guaranteed performance channels, the network must handle dynamic changes in the channel traffic specification [56, 128]; several other researchers have designed mechanisms for supporting these dynamic changes [83, 32]. It will be useful and interesting to extend these mechanisms to support dynamic changes in group traffic specification.

The simulation experiments used very simple, synthetic workloads; it would be interesting to see the results of experiments with more realistic, complex workload models. In particular, the simulations of Chapter 4 assumed a model in which the advance requests were for conferences and the larger the conference, the longer the advance notice period. We will gain useful knowledge by running experiments with different workload models. Collecting realistic workload traces and understanding the relationships between traffic model and real data are important areas to be explored.

It will be useful to characterize the user behavior when a resource request is *rejected.* For conference channels, establishment relationships [63] may exist among the conference channels; for example, if user A is not getting the audio channel from user B, then, at least in some scenarios, A will not be interested in seeing the video from B either. In this case, *rejecting* the audio from B leads to the client rejecting the video as well, thereby reducing the network resource allocation. Another scenario is that the users will repeat the reservation request; this can lead to congestion with many users repeating requests that can not be fulfilled. It is nor clear how the RCAP protocols can handle these situations.

The simulation models chose a simple, relatively homogeneous workload model; this permitted us to reasonably compare the system performance under various conditions. It would be useful to run simulations with more heterogeneous load distribution, especially with different receivers requesting different QoS. To evaluate the performance under such models, we will first need to define appropriate price functions that reflect the revenues that the network can expect for setting up those connections.

In the near future, we can expect the emergence of small network service providers with bandwidth leased from telcos and other communication infrastructure owners (using resource partitioning, for example); these network providers will be able to quickly change their network size by making appropriate requests to the telcos. This will have some interesting implications on routing, network provisioning, and service pricing. For example, if the telcos support unlimited, penalty-free changes, these network service providers will only request the resources that they need. They will also select routes based solely on the prices and the telcos will have to dynamically adapt the prices to the demand and availability of resources. The interactions of these mechanisms will provide many interesting and challenging problems.

When resources allocated at a network server (under resource partitioning) get close to the total resource available at that server ("real-time resource congestion"), the network system has several options: (a) the server can request an increase in resource allocation; (b) the routing system can route channels away from that server; (c) the network can re-route some of the existing channels away from that server; (d) the network can dynamically migrate resources using techniques similar to those described in [8]; or (e) with media-scaling, request the channels passing through that server to reduce their data rates. It will be interesting and useful to study the interaction of these diverse mechanisms and to investigate the policies for selecting among these options.

# Bibliography

[1] Tony Ballardine, Paul Francis, and Jon Crowcroft. Core Based Trees (CBT): an architecture for scalable inter-domain multicast routing. In *Proceedings of SIGCOMM 93*, San Francisco, CA, September 1993.

[2] Anindo Banerjea, Domenico Ferrari, Bruce Mah, Mark Moran, Dinesh Verma, and Hui Zhang. The Tenet real-time protocol suite: Design, implementation, and experiences. Technical Report TR-94-059, International Computer Science Institute, Berkeley, California, November 1994. Also to appear in IEEE/ACM Transactions on Networking, 1995.

[3] Anindo Banerjea and Bruce Mah. The design of a real-time channel administration protocol, June 1991. Internal technical report.

[4] Anindo Banerjea and Bruce Mah. The real-time channel administration protocol. In *Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 160–170, Heidelberg, Germany, November 1991. Springer-Verlag.

[5] Arthur W. Berger, Samuel P. Morgan, and Amy R. Reibman. Statistical multiplexing of layered video streams over ATM networks with leaky-bucket traffic descriptors, 1993. preprint.

[6] Steven Berson and Daniel Zappala. Looping and wildcard filters. "Pre-print", March 1995.

[7] Riccardo Bettati, Domenico Ferrari, Amit Gupta, Wendy Heffner, Wingwai Howe, Quyen Nguyen, Mark Moran, and Raj Yavatkar. Connection establishment for multi-party real-time communication. In *Proceedings of Fifth International Workshop on Network and Operating Systems Support for Distributed Audio and Video*, Durham, NH, April 1994.

[8] Riccardo Bettati and Amit Gupta. Dynamic resource migration for multi-party real-time communication. Technical Report TR-95-060, International Computer Science Institute, Berkeley, California, October 1995.

[9] Robert Braden, David Clark, and Scott Shenker. Integrated services in the internet architecture: an overview. Request for Comments (Informational) RFC 1633, Internet Engineering Task Force, June 1994.

[10] Robert Braden and Lixia Zhang. Minutes of the RSVP Working Group. In *IETF meeting*, Danvers, NH, April 1995.

[11] CCITT proposed recommendation i.311, June 1991.

[12] Navin Chaddha, Mohan Vishwanath, and Philip A. Chou. Hierarchical vector quantization of perceptually weighted block transforms. In *Proceedings of the Data Compression Conference*, Snowbird, UT, 1995.

[13] Jolly Chen, Ray Larson, and Michael Stonebraker. The Sequoia 2000 object browser. Technical Report S2K-91-04, Sequoia 2000 project, University of California at Berkeley, 1991.

[14] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Math. Stat.*, 23:493–509, 1952.

[15] Israel Cidon, Jeff Derby, Inder Gopal, and Bharath Kadaba. A critique of ATM from a data communication perspective. *Journal of High Speed Networking*, 1(2), March 1993.

[16] David Clark. The design philosophy of the DARPA internet protocols. In *Proceedings of ACM SIGCOMM'88*, pages 106–114, Stanford, CA, August 1988.

[17] David Clark. Policy routing in Internet protocols, May 1989. RFC 1102, SRI Network Information Center.

[18] David Clark, Scott Shenker, and Lixia Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. In *Proceedings of ACM SIGCOMM'92*, pages 14–26, Baltimore, Maryland, August 1992.

[19] Rene L. Cruz. A calculus for network delay, part I : Network elements in isolation. *IEEE Transaction of Information Theory*, 37(1):114–121, 1991.

[20] Ying dar Lin, Tzu chieh Tsai, San chiao Huang, and Mario Gerla. HAP: a new model for packet arrivals. In *Proceedings of SIGCOMM 93*, San Francisco, CA, September 1993.

[21] S. Deering, D. Estrin, D. Farinacci, B. Fenner, V. Jacobson, and A. Helmy. Interoperability architecture and mechanisms for PIM-SM. *Internet Draft*, June 1995.

[22] S. Deering, D. Estrin, D. Farinacci, and V. Jacobson. Protocol independent multicast (PIM), dense mode protocol : Specification. *Internet Draft*, March 1994.

[23] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei. Protocol independent multicast (PIM), sparse mode protocol : Specification. *Internet Draft*, March 1994.

[24] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, L. Wei, P. Sharma, and A. Helmy. Protocol independent multicast (PIM) : Motivation and architecture. *Internet Draft*, May 1995.

[25] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, L. Wei, P. Sharma, and A. Helmy. Protocol independent multicast (PIM): Specification. *Working Draft*, June 1995.

[26] Stephen Deering, Deborah Estrin, Dino Farinacci, and Van Jacobson. An architecture for wide-area multicast routing. In *Proceedings of SIGCOMM '94*, University College London, London, U.K., September 1994. ACM.

[27] Stephen E. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, December 1991.

[28] Steve Deering. *Host Extensions for IP Multicasting*. ARPANET Working Group Requests for Comment, DDN Network Information Center, SRI International, Menlo Park, CA, August 1989. RFC-1112.

[29] Steve Deering. Internet multicast routing: State of the art and open research issues, October 1993. MICE Seminar.

[30] Mikael Degermark, Torsten Kohler, Stephen Pink, and Olov Schelen. Advance reservations for predictive service. In *Proceedings of Fifth International Workshop on Network and Operating Systems Support for Distributed Audio and Video*, Durham, NH, April 1995.

[31] L. Delgrossi and L. Berger. Internet stream protocol version 2 (ST2) protocol specification - version ST2+. Request for Comments (Standard) RFC 1819, Internet Engineering Task Force, August 1995.

[32] Luca Delgrossi, Christian Halstrick, Dietmar Hehmann, Ralf Guido Herrtwich, Oliver Krone, Jochen Sandvoss, and Carsten Vogt. Media scaling with HeiTS. IBM European Networking Center, Heidelberg, Germany, March 1993.

[33] Luca Delgrossi, Christian Halstrick, Ralf Guido Herrtwich, and Heinrich Stüttgen. HeiTP: a transport protocol for ST-II. In *Proceedings of GLOBECOMM*, pages 1369–1373 (40.02), Orlando, Florida, December 1992. IEEE.

[34] Luca Delgrossi, Ralf Guido Herrtwich, , Frank Oliver Hoffmann, and Sibylle Schaller. Receiver-initiated communication with ST-II. *Multimedia Systems*, 2(4):141–149, October 1994.

[35] Luca Delgrossi, Ralf Guido Herrtwich, and Frank Oliver Hoffmann. An implementation of ST-II for the Heidelberg transport system. *Journal of Internetworking Research and Experience*, September 1993.

[36] Luca Delgrossi, Ralf Guido Herrtwich, Carsten Vogt, and Lars C. Wolf. Reservation protocols for internetworks: A comparison of ST-II and RSVP. In *Proceedings of the Fourth International Workshop on Network and OS Support for Digital Audio and Video*, Lancaster, U.K., November 1993. ACM.

[37] Luca Delgrossi, Sibylle Schaller, Hartmut Wittig, and Lars Wolf. Issues of reserving resources in advance. In *Proceedings of Fifth International Workshop on Network and Operating Systems Support for Distributed Audio and Video*, Durham, NH, April 1995.

[38] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simulation of a fair queueing algorithm. In *Journal of Internetworking Research and Experience*, pages 3–26, October 1990. Also in Proceedings of ACM SIGCOMM'89, pp 3-12.

[39] Deborah Estrin. Policy requirements for inter administrative domain routing, Nov 1989. RFC 1125, SRI Network Information Center.

[40] Domenico Ferrari. Real-time communication in packet-switching wide-area networks. Technical Report TR-89-022, International Computer Science Institute, Berkeley, California, May 1989.

[41] Domenico Ferrari. Client requirements for real-time communication services. *IEEE Communications Magazine*, 28(11):65–72, November 1990.

[42] Domenico Ferrari. Design and applications of a delay jitter control scheme for packet-switching internetworks. In *Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 72–83, Heidelberg, Germany, November 1991. Springer-Verlag. Also in *Computer Communications* 15(6):367-373, July-August 1992.

[43] Domenico Ferrari. Real-time communication in an internetwork. *Journal of High Speed Networks*, 1(1):79–103, 1992.

[44] Domenico Ferrari, Anindo Banerjea, and Hui Zhang. Network support for multimedia: a discussion of the Tenet approach. *Computer Networks and ISDN Systems*, pages 1267–1280, July 1994.

[45] Domenico Ferrari and Amit Gupta. Resource partitioning in real-time communication. In *Proceedings of IEEE Symposium on Global Data Networking*, pages 128–135, Cairo, Egypt, December 1993.

[46] Domenico Ferrari, Amit Gupta, Mark Moran, and Bernd Wolfinger. A continuous media communication service and its implementation. In *Proceedings of GLOBECOM '92*, Orlando, Florida, December 1992.

[47] Domenico Ferrari, Amit Gupta, and Giorgio Ventre. Distributed advance reservation of real-time connections. In *Proceedings of Fifth International Workshop on Network and Operating Systems Support for Distributed Audio and Video*, Durham, NH, April 1995.

[48] Domenico Ferrari, Joe Pasquale, and George Polyzos. Network issues for Sequoia 2000. In *Proceedings of COMPCOM 92*, pages 401–406, San Francisco, CA, February 1992.

[49] Domenico Ferrari and Dinesh Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, 8(3):368–379, April 1990.

[50] Sally Floyd. Issues in flexible resource management for datagram networks. In *Proceedings of the 3rd Workshop on Very High Speed Networks*, Maryland, March 1992.

[51] Sally Floyd. Link-sharing and resource management models for packet networks. Unpublished, September 1993.

[52] Shivi Fotedar, Mario Gerla, Paolo Crocetti, and Luigi Fratta. ATM virtual private networks. *Communications of the ACM*, pages 101–109, February 1995.

[53] Alexander G. Fraser, Chuck R. Kalmanek, A.E. Kaplan, William T. Marshall, and R.C. Restrick. Xunet2: A nationwide testbed in high-speed networking. In *Proceedings of INFOCOM'92*, Firenze, Italy, May 1992.

[54] Mark W. Garrett and Martin Vetterli. Joint source/channel coding of statistically multiplexed real-time services on packet networks. *IEEE/ACM Transactions on Networking*, 1(1):71–80, February 1993.

[55] M. Ghanbari. Two-layer coding of video signals for VBR networks. *IEEE Journal on Selected Areas in Communications*, 7(5):771–781, June 1989.

[56] Matt Grossglauser, Srinivas Keshav, and D. Tse. The case against variable bit rate service. In *Proceedings of Fifth International Workshop on Network and Operating Systems Support for Distributed Audio and Video*, Durham, NH, April 1995.

[57] Amit Gupta. Real-time communication with mobile hosts - rerouting for handoffs. CS292J Class Report, University of California at Berkeley, May 1993.

[58] Amit Gupta and Domenico Ferrari. Resource partitioning for multi-party real-time communication. Technical Report TR-94-061, International Computer Science Institute, Berkeley, California, November 1994. Also in IEEE/ACM Transactions on Networking, October 1995.

[59] Amit Gupta and Domenico Ferrari. Admission control for advance-reserved real-time connections. In *Proceedings of IEEE HPCS 95*, Mystic, CT, August 1995.

[60] Amit Gupta, Wendy Heffner, Mark Moran, and Clemens Szyperski. Multi-party real-time communication in computer networks. In *Collected abstracts of 4th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 37–39, Lancaster, UK, November 1993.

[61] Amit Gupta, Wingwai Howe, Quyen Nguyen, and Mark Moran. Evaluation of resource sharing beneffits. Technical Report TR-94-051, International Computer Science Institute, Berkeley, California, October 1994.

[62] Amit Gupta, Winnie Howe, Mark Moran, and Quyen Nguyen. Resource sharing in multi-party realtime communication. In *Proceedings of INFOCOM 95*, Boston, MA, April 1995.

[63] Amit Gupta and Mark Moran. Channel groups: A unifying abstraction for specifying inter-stream relationships. Technical Report TR-93-015, International Computer Science Institute, Berkeley, California, March 1993.

[64] Amit Gupta and Kurt Rothermel. Fault handling for multi-party real-time communication. Technical Report TR-95-059, International Computer Science Institute, Berkeley, California, October 1995.

[65] D. Hehmann, R. G. Herrtwich, W. Schultz, T. Schütt, and R. Steinmetz. Implementing HeiTS: Architecture and implementation strategy of the Heidelberg high-speed transport system. In *Proc. Second International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 33–44, Heidelberg, Germany, November 1991.

[66] Ralf Guido Herrtwich and Luca Delgrossi. Beyond ST-II: fulfilling the requirements of multimedia communication. In *Third International Workshop on network and operating system support for digital audio and video*, pages 23–29, San Diego, California, November 1992. IEEE Computer and Communications Societies.

[67] Kelvin K. Y. Ho. Comparative analysis of virtual-circuit routing control for isdn frame-relay networks. In *IEEE Global Telecommunications Conference*, number GlobeCom'90, pages 800.6.1–800.6.5, San Diego, California, December 1990.

[68] Jay M. Hyman, Aurel A. Lazar, and Giovanni Pacifici. Modeling VC, VP, and VN bandwidth assignment strategies in broadband networks. In *Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 99–110, Lancaster, U.K., November 1993. Lancaster University. Lecture Notes in Computer Science 712, Springer Verlag.

[69] Sugih Jamin, Peter Dantzig, Scott Shenker, and Lixia Zhang. A measurement-based admission control algorithm for integrated services packet networks. In *Proceedings of SIGCOMM 95*, Cambridge, MA, August 1995.

[70] Sugih Jamin, Scott Shenker, Lixia Zhang, and David Clark. An admission control algorithm for predictive real-time service. In *Proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 349–356, San Diego, CA, November 1992.

[71] Ahmed Karmuch, Luis Orozco-Barbosa, Nicolas D. Georganas, and Morris Goldberg. A multimedia medical communications system. *IEEE Journal on Selected Areas in Communications*, 8(3):325–339, April 1990.

[72] E. W. Knightly and G. Ventre. Galileo: a tool for simulation and analysis of real-time networks. In *Proceedings of IEEE 1993 International Conference on Network Protocols*, pages 264–271, San Francisco, CA, October 1993.

[73] Jim Kurose. On computing per-session performance bounds in high-speed multi-hop computer networks. In *ACM SigMetrics'92*, 1992.

[74] Jim Kurose. Open issues and challenges in providing quality of service guarantees in high-speed networks. *ACM Computer Communication Review*, 23(1):6–15, January 1993.

[75] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of ACM*, 20(1):46–61, January 1973.

[76] Bruce Mah. A mechanism for administration of real-time channels. Master's thesis, Tech. Report UCB/CSD-93-735, University of California, Berkeley, CA, March 1993.

[77] N.F. Maxemchuck. *Dispersity Routing in Store and Forward Networks.* PhD dissertation, University of Pennsylvania, May 1975.

[78] Steven McCanne and Van Jacobson. *VIC: video conference.* Lawrence Berkeley Laboratory and University of California, Berkeley. Software on-line[2].

[79] Steven McCanne and Van Jacobson. *vic:* a flexible framework for packet video. In *Proceedings of ACM Multimedia '95.* ACM, November 1995.

[80] Steven McCanne and Martin Vetterli. Joint source/channel coding for multicast packet video. *IEEE International Conference on Image Processing*, October 1995.

[81] Danny Mitzel, Deborah Estrin, Scott Shenker, and Lixia Zhang. An architectural comparison of ST-II and RSVP. In *Proceedings of INFOCOM 94*, Toronto, CANADA, June 1994.

[82] Danny Mitzel and Scott Shenker. Asymptotic resource consumption in multicast reservation styles. In *Proceedings of SIGCOMM 94*, London, UK, September 1994.

[83] Rebbie Moon. Dynamic traffic management in Scheme 2 of the Tenet real-time protocols. Master's thesis, University of California, Berkeley, CA, December 1995.

[84] Mark Moran and Riccardo Gusella. System support for efficient dynamically-configurable multi-party interactive multimedia applications. In *Proceedings of Third International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 143–156, San Diego, CA, November 1992.

[85] Rolf Oppliger, Amit Gupta, Mark Moran, and Riccardo Bettati. A security architecture for Tenet Scheme 2. Technical Report TR-95-051, International Computer Science Institute, Berkeley, California, August 1995.

[2]ftp://ftp.ee.lbl.gov/conferencing/vic

[86] Antonio Ortega and Martin Vetterli. Multiple leaky buckets for increased statistical multiplexing of ATM video. In *Proceedngs of Packet Video Workshop*, Portland, OR, September 1994.

[87] Pramod Pancha and Magda El Zarki. A look at the MPEG video coding standard for variable bit rate video transmission. *Proc. IEEE INFOCOM '92*, May 1992.

[88] Abhay Kumar J. Parekh. *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks.* PhD dissertation, Massachusetts Institute of Technology, February 1992.

[89] Abhay Kumar J. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control - the single node case. In *Proceedings of the INFOCOM'92*, 1992.

[90] Abhay Kumar J. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The multiple node case. In *Proceedings of the INFOCOM'93*, pages 521–530, San Francisco, CA, March 1993.

[91] Colin Parris. *Dynamic connection management for real-time networks.* PhD dissertation, University of California at Berkeley, August 1994.

[92] Colin Parris and Domenico Ferrari. A dynamic connection management scheme for guaranteed performance services in packet-switching integrated services networks. Technical Report TR-93-005, International Computer Science Institute, Berkeley, California, January 1993.

[93] Colin Parris, Giorgio Ventre, and Hui Zhang. Graceful adaptation of guaranteed performance service connections. In *Proceedings of IEEE GLOBECOM'93*, Houston, TX, November 1993.

[94] Colin Parris, Hui Zhang, and Domenico Ferrari. Dynamic management of guaranteed performance multimedia connections, April 1993. To appear in ACM Journal of Multimedia Systems.

[95] Craig Partridge. Isochronous applications do not require jitter-controlled networks, September 1991. RFC 1157.

[96] Craig Partridge. A proposed flow specification. Request for Comments (Informational) RFC 1363, Internet Engineering Task Force, September 1992.

[97] Craig Partridge and Stephen Pink. An implementation of the revised internet stream protocol (ST-2). In *Journal of Internetworking Research and Experience*, pages 27–54, 1992.

[98] Joseph Pasquale, George Polyzos, Eric Anderson, and Vachaspati Kompella. The multimedia multicast channel. In *Proceedings of Third International Workshop on Network and Operating Systems Support for Distributed Audio and Video*, San Diego, CA, November 1992.

[99] Jean Ramaekers and Giorgio Ventre. Client-network interaction in a real-time communication environment. In *Proceedings of GLOBECOMM '92*, pages 1140–1144, Orlando, Florida, December 1992.

[100] Jean Ramaekers and Giorgio Ventre. Quality-of-service negotiation in a real-time communication network. Technical Report TR-92-023, International Computer Science Institute, Berkeley, California, April 1992.

[101] Kannan Ramchandran, Antonio Ortega, K. Metin Uz, and Martin Vetterli. Multiresolution broadcast for digital HDTV using joint source/channel coding. *IEEE Journal on Selected Areas in Communications*, 11(1):6–23, January 1993.

[102] Wilko Reinhardt. Advance reservation of network resources for multimedia applications. In *Proceedings of ICAWA 94*, Germany, October 1994.

[103] Ken-Ichi Sato, Satoru Ohta, and Ikuo Tokizawa. Broadband ATM network architecture based on virtual paths. *IEEE transactions on communications*, pages 1212–1222, August 1990.

[104] J. M. Schneider, T. Preuß, and P. S. Nielsen. Management of virtual private networks for integrated broadband communications. In Deepinder P. Sidhu, editor, *Proceedings of SIGCOMM 1993*, pages 224–237, San Francisco, California, September 1993. ACM. also in *Computer Communication Review* 23 (4), Oct. 1992.

[105] Henning Schulzrinne. Dynamic configuration of conferencing applications using pattern-matching multicast. In *Proceedings of the Fifth International Workshop on Network and OS Support for Digital Audio and Video*, Durham, NH, April 1995. ACM.

[106] Karen Seo. ST-II – new release, November 1991. Connection IP (cip) mailing list.

[107] Jerome M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3462, December 1993.

[108] Scott Shenker and Lee Breslau. Two aspects of reservation establishment. In *Proceedings of SIGCOMM 95*, Cambridge, MA, August 1995.

[109] Michael Stonebraker. An overview of the Sequoia 2000 project. In *Proceedings of COMPCOM 92*, San Francisco, CA, February 1992.

[110] Michael Stonebraker, Jolly Chen, Nobuko Jathan, Caroline Paxson, and Jiang Wu. Tioga: Providing data management support for scientific visualization applications. In *Proceedings of the 19th VLDB Conference*, Dublin, Ireland, 1993.

[111] Fore Systems, June 1994. ATM Switch Specification.

[112] Clemens Szyperski and Giorgio Ventre. A characterization of multi-party interactive multimedia applications. In *High Performance Network Research Report*, January 1993.

[113] David Taubman and Avideh Zakhor. Multi-rate 3-D subband coding of video. *IEEE Transactions on Image Processing*, 3(5):572–588, September 1994.

[114] Claudio Topolcic. Experimental internet stream protocol, version 2 (ST-II), October 1990. RFC 1190.

[115] Thierry Turletti and Jean-Chrysostome Bolot. Issues with multicast video distribution in heterogeneous packet networks. In *Proceedings of the Sixth International Workshop on Packet Video*, Portland, OR, September 1994.

[116] Jonathan Turner. New directions in communications(or which way to the information age?). *IEEE Communication Magazine*, 24(10), October 1986.

[117] Dinesh Verma. *Guaranteed Performance Communication in High Speed Networks.* PhD dissertation, University of California at Berkeley, November 1991.

[118] Dinesh Verma and Hui Zhang. Design documents for RMTP/RTIP, May 1991. Unpublished internal technical report.

[119] Mohan Vishwanath and Phil Chou. An efficient algorithm for hierarchical compression of video. *IEEE International Conference on Image Processing*, November 1994.

[120] D. Waitzman, C. Partridge, and S. Deering. *Distance Vector Multicast Routing Protocol.* Arpanet Working Group Requests for Comment, DDN Network Information Center, SRI International, Menlo Park, CA, November 1988. RFC-1075.

[121] Ian Wakeman, Atanu Ghosh, Jon Crowcroft, Sally Floyd, and Van Jacobson. Implementing real-time packet forwarding policies using streams. In *Proceedings of USENIX 1995 Technical Conference*, Nashville, TN, January 1995.

[122] L. Wei and D. Estrin. The trade-offs of multicast trees and algorithms. In *Proceedings of the 1994 international conference on computer communications and networks*, San Francisco, September 1994.

[123] Ron Widyono. The design and evaluation of routing algorithms for real-time channels. Technical Report TR-94-024, International Computer Science Institute, Berkeley, California, June 1994.

[124] Bernd Wolfinger and Mark Moran. A continuous media data transport service and protocol for real-time communication in high speed networks. In *Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 171–182, Heidelberg, Germany, November 1991. Springer-Verlag.

[125] Hui Zhang and Domenico Ferrari. Rate-controlled static priority queueing. In *Proceedings of IEEE INFOCOM'93*, pages 227–236, San Francisco, California, April 1993.

[126] Hui Zhang and Tom Fisher. Preliminary measurement of RMTP/RTIP. In *Proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video*, San Diego, CA, November 1992. Springer-Verlag.

[127] Hui Zhang and Srinivasan Keshav. Comparison of rate-based service disciplines. In *Proceedings of ACM SIGCOMM'91*, pages 113–122, Zurich, Switzerland, September 1991.

[128] Hui Zhang and Ed Knightly. A new approach to support delay sensitive VBR video in packet switched networks. In *Proceedings of Fifth International Workshop on Network and Operating Systems Support for Distributed Audio and Video*, Durham, NH, April 1995.

[129] Hui Zhang, Dinesh Verma, and Domenico Ferrari. Design and implementation of the real-time internet protocol. In *Proceedings of the IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*, Tucson, Arizona, February 1992.

[130] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. RSVP: A new resource reservation protocol. *IEEE Networks Magazine*, 31(9):8–18, September 1993.