INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. • Suite 600 • Berkeley, California 94704-1198 • (510) 643-9153 • FAX (510) 643-7684

Distributed advance reservation of real-time connections

Domenico Ferrari, Amit Gupta, Giorgio Ventre*
E-mail: {ferrari,amit,ventre}@icsi.berkeley.edu

Tenet Group
University of California at Berkeley, and
International Computer Science Institute

TR-95-008

March 1995

Abstract

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. This paper discusses the requirements of the clients of an advance reservation service, and a distributed design for such a service. The design is described within the context of the Tenet Real-Time Protocol Suite 2, a suite being developed for multi-party communication, which will offer advance reservation capabilities to its clients based on the principles and the mechanisms proposed in the paper. Simulation results providing useful data about the performance and some of the properties of these mechanisms are also presented. We conclude that the one described here is 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.

*Now with Dipartimento di Informatica e Sistemistica, Università degli Studi di Napoli "Federico II", Napoli, Italy

1 Introduction

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 [1, 3, 4, 14, 15, 16]). Let us consider, for example, the Tenet scheme [8], which has been one of the first to be implemented (in the Tenet Real-Time Protocol Suite 1 [1]). The Tenet protocols can coexist with non-real-time protocols such as those in the Internet suite. A network or internetwork in which they run will be able to carry integrated real-time and non-real-time traffic; however, the version of the scheme that has been described and implemented allows the real-time service to establish real-time channels to saturate (in the worst case), and, when channels with statistical delay or reliability bounds are involved, even to overbook, some of the network's resources. Since the environment is a packet-switching one, non-real-time traffic is able, even when resources are fully booked or overbooked, to exploit any resources not currently in use (even if they have been reserved) by real-time connections. In the client-service interface of the Tenet Suite 1 [1], 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, 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.

This paper addresses 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.

Section 2 discusses the service requirements for advance reservations. In Section 3,

we describe the distributed advance reservations mechanisms we have designed for, and are implementing in, the Tenet Suite 2 [10]. The principles on which our mechanisms are based, however, are easily portable to other approaches and protocols for real-time communication. We also present several simulation results in Section 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.

The service model in the existing proposals and realizations of real-time communication services, including that in the Tenet Suite 1 [1], 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. The network, incidentally, would not have much use for this information, since planning future allocations would require predicting arrivals of new requests along with departures of currently existing channels, and arrivals cannot be precisely predicted in an environment in which advance reservations are not supported. The current establishment model, in which channels are to be created immediately (i.e., as soon as possible), coincides with that of a normal telephone call, whose expected duration never has to be specified by the caller.

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.

3 An advance reservations service

In this section, we describe the design of an advance reservation service for a real-time (or integrated-services) network that uses the Tenet protocols. The service will be offered by the Tenet Suite 2, but could be easily implemented also in the Tenet Suite 1. The basic design alternatives and decisions are discussed in Section 3.1. In Section 3.2, we provide some background information about the Tenet Scheme 2, with the objective of facilitating the understanding of our design. Section 3.3 is devoted to a description of the distributed mechanisms we have chosen to accomplish our goals. While this description is presented in the framework of the Tenet protocols, the underlying ideas and techniques are also applicable to other schemes and protocols.

3.1 Design alternatives and decisions

Since clients are expected to accept rather easily the requirement that advance channels be created for a definite amount of time, we have chosen to enforce this requirement in our design due to its expected beneficial effect on the utilization of network resources. The question may be asked, then, whether the specification of a duration should be required also for immediate channels. The advantages of such a decision would be uniformity and the potential for better resource management; the main disadvantage would be that clients who have an immediate need for transferring information do not often know how long the session will be, and, in any case, are not accustomed to having to specify its duration. This disadvantage is a very serious one, so serious that we believe it cannot be ignored. In our design, immediate channels will normally be created with an indefinite duration.

In the establishment of immediate channels, which in the scheme described in [8] only considers the situation at the time the request is made, we must now look at all the future situations as well: Figure 1 shows a case in which no immediate channel can be created through a server (i.e., a network component that has resources to be allocated) at time t_1 even though the resource in question is fully available in the server at that time. This complication can be avoided, and the establishment of immediate channels still kept as fast as possible, by separating the two types of channels so that the admission tests for a new channel only take into account the channels of its type in each server. Resource partitioning [7, 9], a service the Tenet Suite 2 will offer network managers, is an almost perfect solution for this problem. The *immediate partition* treats any new request exactly as described in [8]. The *advance partition* must instead use a different mechanism (to be presented in Section 3.3) to test new requests for admission. Note that this mechanism can be made more complicated, if necessary, than the one used for immediate channels, since advance reservation requests are usually less urgent than immediate requests, and can be run in the background wherever they are run. Also, the two types of requests do not have to be processed serially or in a specific order by a server to reflect their relative importance; being totally independent of each other, they may be executed simultaneously and in any order.

The partition-based solution raises, however, a problem: each resource has to be allocated statically to each partition, and this may cause inefficiencies due to fragmentation [9] and, worse, to poor allocation decisions; one of the two partitions for a resource may be saturated while the other is almost empty. Since choosing *a priori* allocations suitable for the types of requests that will arrive is hard, the implementation of a movable boundary between the two partitions must be explored; mechanisms for obtaining this result will be described in Section 3.3.

With the introduction of the two partitions referred to above, we do not prevent other uses of resource partitioning, as long as the additional partitions to be created can be regarded as sub-partitions of the immediate or advance ones. The only restriction of this arrangement is that, for example, an advance channel cannot be a

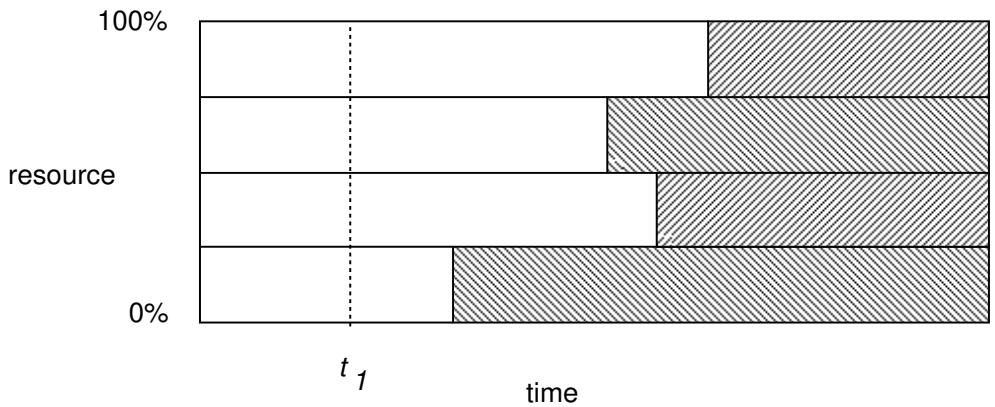

Figure 1: A request for an immediate channel cannot be accepted at time t_1, even though the resource is 100% free at that time. The shaded areas represent fractions of the resource that have been reserved in advance.

member of a sub-partition of the immediate partition, or vice versa.

As for the minimum advance notice required for an advance reservation, which was mentioned in Section 2, it seems reasonable to set it to zero: if there is enough room for the requested duration in all the servers to be traversed within the advance partition, why should the network not accept such an "immediate reservation"? This argument suggests that the crucial distinction between the two partitions might be not so much that between advance and immediate channels, but that between channels of definite and indefinite duration. In fact, it would perhaps be useful to allow for the advance reservation of indefinite-duration channels, but in our design, for simplicity, we ignore this possibility, and do not allow advance reservations of indefinite-duration requests. Thus, all such requests are immediate, and belong in the immediate partition, while all definite-duration requests are tested against the advance partition, even if their starting time coincides with the current time.

Finally, a decision is to be made concerning 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 is described in Section 3.3.

3.2 The Tenet Scheme 2

The scheme on which the Tenet Suite 2 is based implements the *multicast real-time channel* abstraction. This communication abstraction is defined as a simplex connection between a source and a *target set*, i.e., a set of destinations, capable of guaranteeing a given (and possibly different) quality of service (QoS) to each destination in the target set. A real-time channel ("channel" for short in the sequel) is characterized in the Tenet schemes at the network layer by a quadruple of traffic parameters and, for each destination, a quadruple of QoS parameters; these two quadruples are listed in Table 1. Scheme 2 allows channels to be established from the source or from the destinations; we describe here, for brevity, only the former procedure.

When a client wants to set up a channel, it invokes the Real-Time Channel Administration Protocol (RCAP) and passes to it suitable identifiers for the source and the destinations, as well as the source's traffic parameters and each destination's QoS parameters. RCAP gets a possible route for the channel from a Routing Server, and issues a *channel-establish* message from the source. This message follows the given route, replicating itself at each subtree vertex it encounters on its path, and sending a copy of the message down each of the branches of that subtree. Admission tests are performed at each server (where a server is a node or a link) reached by a copy of the message; if any of the tests is unsuccessful, a *channel-reject* message is sent back toward the source; if all tests are successful, a copy of the message is sent to the next server on that path, until it reaches one of the destinations. Each destination makes a final decision about the setup request for the new channel, and returns an *channel-accept* or *channel-reject* message to the source. Messages of both types wait for those from the destinations in the same subtree at the subtree's vertex, where they are merged with them before the resulting message is forwarded on the reverse path toward the source.

Some resources had been tentatively reserved by the *channel-establish* message or one of its replicas in each traversed server. When receiving the returning message(s), which will generally contain both accepts and rejects from the various destinations in the corresponding subtree, the server cancels those reservations (if all the replies are of the *channel-reject* type) or adjusts them according to the information received from each accepting destination. These adjustments are reflected in the return message forwarded to the server immediately upstream, as well as in the amounts of resources actually reserved in the server for the new channel. At the end of all this activity, the source receives the results of the request within a single return message, and transmits them to the client for evaluation and further action.

The channel so created, reaching the destinations that have accepted the request, is now ready for immediate use and remains in existence until it is torn down explicitly

Traffic parameters (of a source)	
Xmin	minimum interpacket interval
Xave	minimum average interpacket interval
I	averaging interval
Smax	maximum packet size

QoS parameters (of a destination)	
Dmax	end-to-end delay bound
Zmin	minimum probability of timely delivery
Jmax	end-to-end delay jitter bound
Wmin	minimum probability of no loss due to buffer overflow

Table 1: Network layer traffic and QoS parameters in the Tenet schemes

by the client or destroyed by an irrecoverable failure.

3.3 A distributed advance reservation mechanism

In a distributed approach, the advance reservation information must be stored in the servers 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 paper, and its discussion is therefore postponed to a future publication.

Having divided each resource in a server into at least two partitions, we can just concern ourselves with the amount of each that is allocated to the advance partition. Since the boundary between the two partitions is movable, we allow this amount to vary from time to time; however, we subdivide 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 advance 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.

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 2: An example interval table in a server

Property (ii) could be regarded as a consequence of property (i), as it is reasonable not to move the boundary between the two partitions when nothing changes; it is only when a channel request that cannot fit in its natural partition comes to life that we may have to use resources from the other partition to support it.

The basic mechanism used to manage the advance partition in a server is the *interval table*, which lists all the advance channels that will traverse the server during a future interval, together with the requirements for each of the server's resources. The interval table, an example of which is shown in Table 2, 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.

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 [7]). 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 advance partition in a server is empty, there is only one interval table; its top row is empty, its start time is the current time (as we have decided not to require any minimum advance notice), and its end time is "infinity". When an advance channel request is received from a client, the source[1] 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 our server causes the only existing interval to be subdivided into three intervals: (current, start), (start, end), and (end, current + max advance notice). 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

[1]The Tenet suites allow receiver-initiated as well as sender-initiated channel establishment. We describe only the sender-initiated procedure here to simplify the discussion.

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 2, tables T_{01} and T_{56} will not be affected by the addition of the new channel, while T_{12} will be relabelled $T_{11'}$ (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.

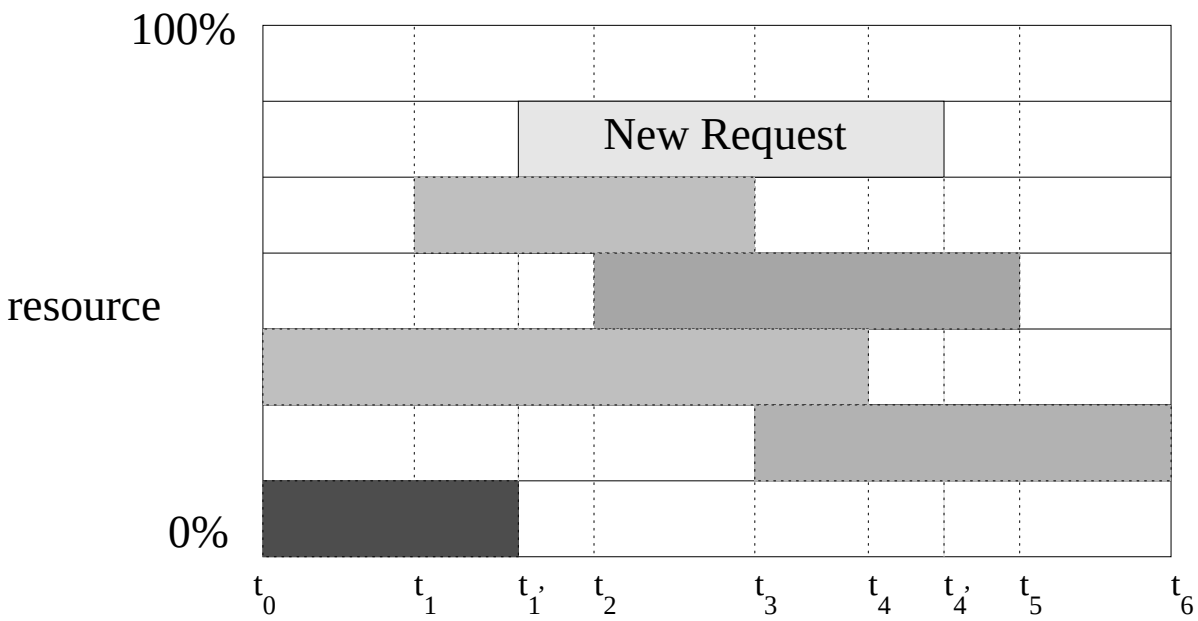

Figure 2: Effects on the intervals and interval tables of the addition of an advance channel

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 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, 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.

We shall now sketch mechanisms for moving the boundary between the two partitions, as mentioned in Section 3.1. First, suppose that in a server one of the interval tables is full when a new advance request is received. Instead of sending a negative return message back towards the source, we may choose to reduce by the needed amount the current allocation of that resource to the immediate partition. This reduction could be *temporary*, i.e., from the current time to the end time of the interval in which the advance partition was saturated, in which case the relevant "Resources available" entry in each of the tables chronologically preceding the saturated one and in the saturated one would have to be increased by the same amount; or the reduction could be *permanent*, in which case all tables in the server would be updated. Of course, if the boundary is moved only temporarily, we have to make sure that the allocation of the resource to the immediate partition is re-increased at the time the table that was saturated is deleted. As long as there is a mechanism that can move the boundary in the other direction on demand, such as the one to be described in the next paragraph, we may be tempted to favor a permanent reduction; however, it may be necessary to put curbs to the expansion of the advance partition, since advance reservations might sometimes be too aggressive and leave too little room to the immediate ones.

Conversely, let a request for an immediate channel be received by a server in which the immediate partition is out of one of the resources. Then, instead of rejecting the channel, the server could look at the interval tables kept by the advance partition to determine whether a sufficient amount of that resource is available there. If such an amount is available in all of the tables, the server can allocate that amount to the immediate partition by reducing the appropriate "Resources available" entries accordingly. When amounts of that resource are released by immediate channels, the server has the option of returning those amounts to the advance partition by suitably updating the tables, or it may do so only on demand, using the mechanism described in the previous paragraph. If, on the other hand, a table is found in which the required amount is not available, the server may reject the request or accept it for a limited time. The latter option introduces a third type of channel, the *immediate channel with definite duration*; however, this is not a channel that may be requested by a client (who can request an advance channel with an immediate start time in-

stead), but a restriction imposed by the network on an immediate channel for lack of resources. We favor this option, and have included it in our design, since we want to maximize network utilization and minimize the blocking probability. A watermark to protect the advance partition may be necessary too, and network managers may find it useful to create only immediate channels with definite duration once this watermark is exceeded, so as to leave room for farther-future reservations while using near-future resources for requested immediate channels instead of keeping them for unlikely advance requests.

All the dilemmas we have mentioned above without resolving them are policy choices. We have designed mechanisms that allow network managers to specify the policies. The evaluation of the possible policies is a topic for future research.

4 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 simple channel requests, with and without the advance reservation mechanisms. To distinguish the effects of the two components of the advance reservation mechanisms, namely partitioning-induced protection and priority changing (which we describe below), we repeated the previous experiments using only one of the two components. The third set of experiments considered multi-party advance reservations 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.

4.1 Simulator description

We ran these simulations on an enhanced version of *Galileo* [13], 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 [10]. For example, we used the NSFNET backbone network topology in our simulations (see Figure 3). We assumed 45Mbps for each link, and we set the propagation delay along the diameter to 40 ms. 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

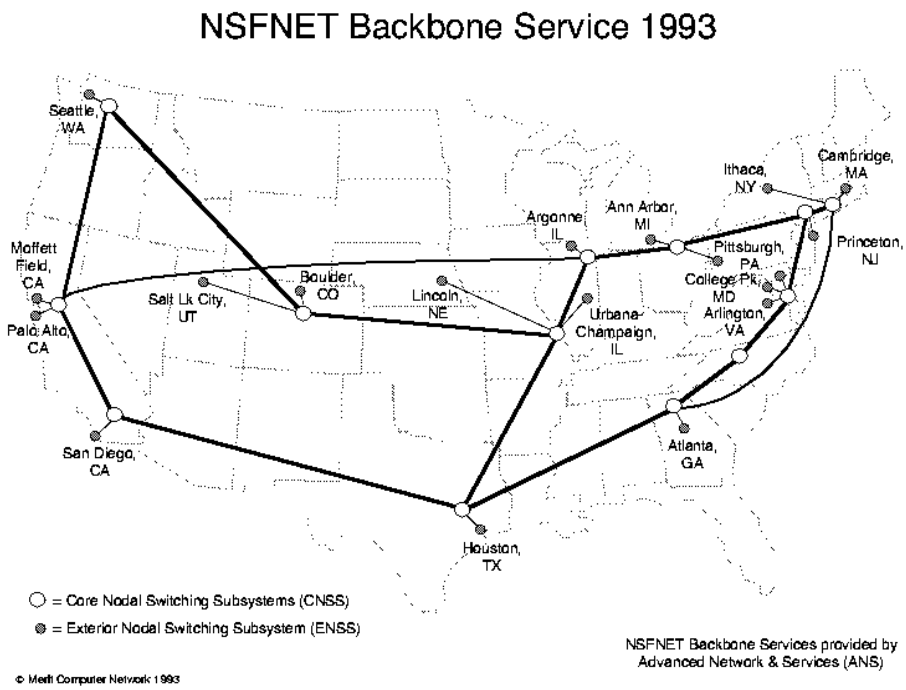

Figure 3: The NSFNET topology

discipline that we used in our simulations). Thus, bandwidth or processing power was the limiting resource in all servers and all scenarios.

4.2 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 duration 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:

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

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 used the EDD scheduling discipline [8], in which admission control tests at a server take $O(n)$ time, where n is the number of resource allocations existing at that server. We summed the values of n (the number of existing channels when a new request arrives at a server) at *all* servers on the new channel's path to obtain an index for the total establishment overhead. This led us to the following metric for comparing computational overheads:

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

4.3 Simulation experiments

Simple workload

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

- 150 simplex unicast connections, and 50 10-person conferences (each conference requiring 10 multicast 9-destination channels), all in the same partition, which was allocated 80% of the network's bandwidth; we call this the *without advance reservations* case; and
- 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; we call this the *advance reservations* case.

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 [11, 12].

In Figure 4 and all of the remaining figures, 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.

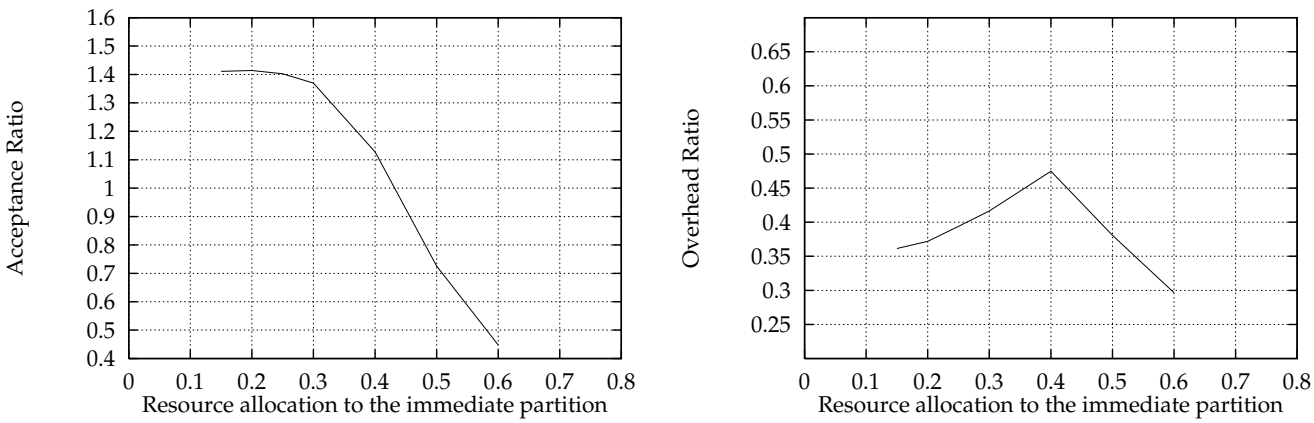

Figure 4: Acceptance ratio and overhead ratio for the first set of experiments

In Figure 4, 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.

The same figure 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 [11]; the overhead increases because we accept more individual channels at the expense of conference channels). However, after reaching a peak value, the overhead start 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). Our measurements show that the admission tests overhead is usually less than 1 ms per server on a 12 SPECmarks (SPEC 92int) SUN SS IPC [2].

Isolating the effects of the 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:

- 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; we call this the *only partitioning* case.

As the graphs of Figure 5 show, the gains (and losses) observed in Figure 4 arise primarily from the isolation and protection provided by the resource partitioning mechanisms. 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 5, the overhead of admission control is substantially lower in the *only partitioning* case than in the *advance reservations* case, where multiple tables are manipulated.

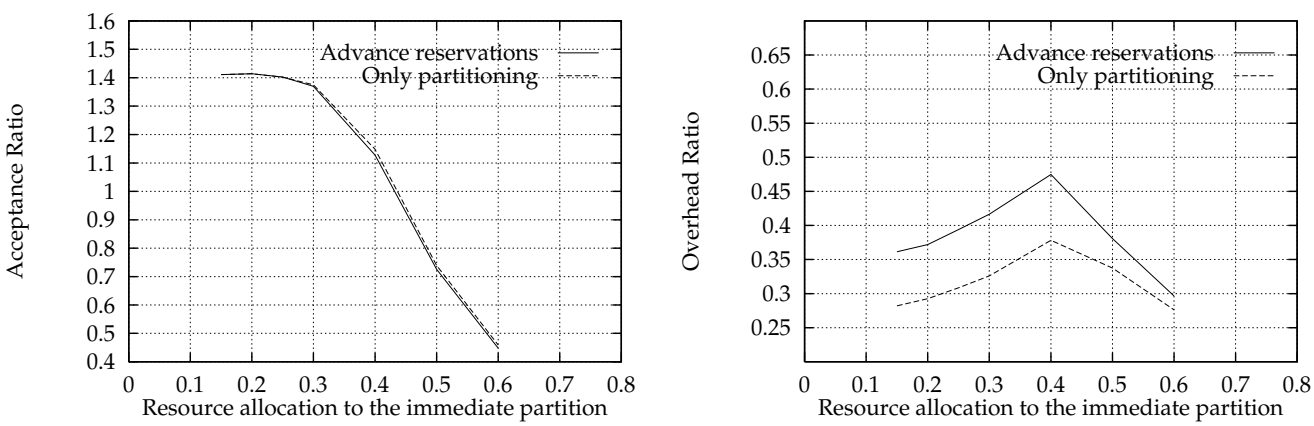

Figure 5: Isolating the effects of the two components

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.

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.

The average advance notice period was set to 3 hours for 10-person conferences, and 6 hours for 50-person conferences. In each case, the actual advance notice period was randomly selected from a uniform distribution around the mean.

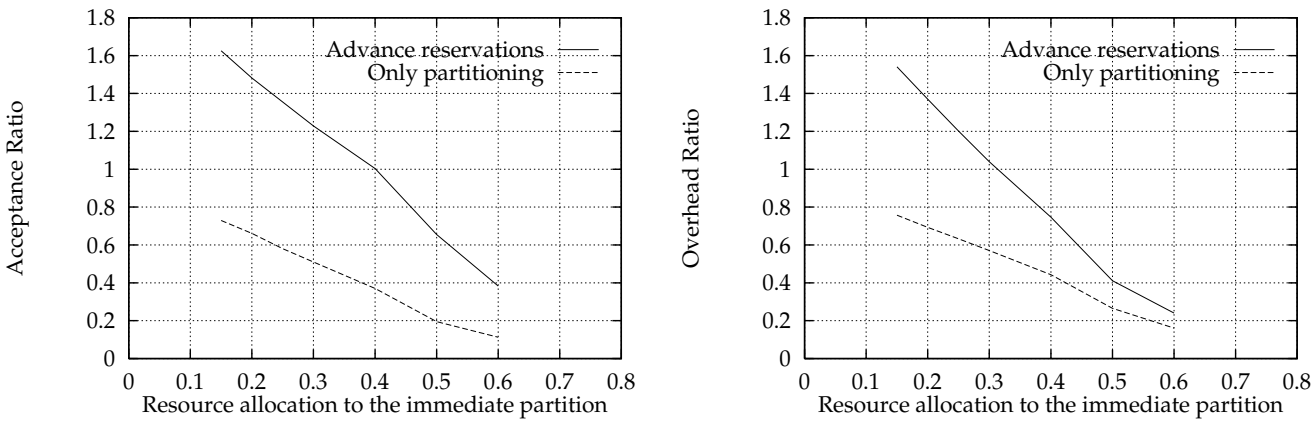

Figure 6: Multiple advance notice periods and conference sizes

As the graphs in Figure 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 imtermediate value of the relative resource allocation.

Effect of time granularity

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

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 *natural* tendency of the users would be to ask for connections with start times and durations that vary randomly and uniformly within specified time intervals.

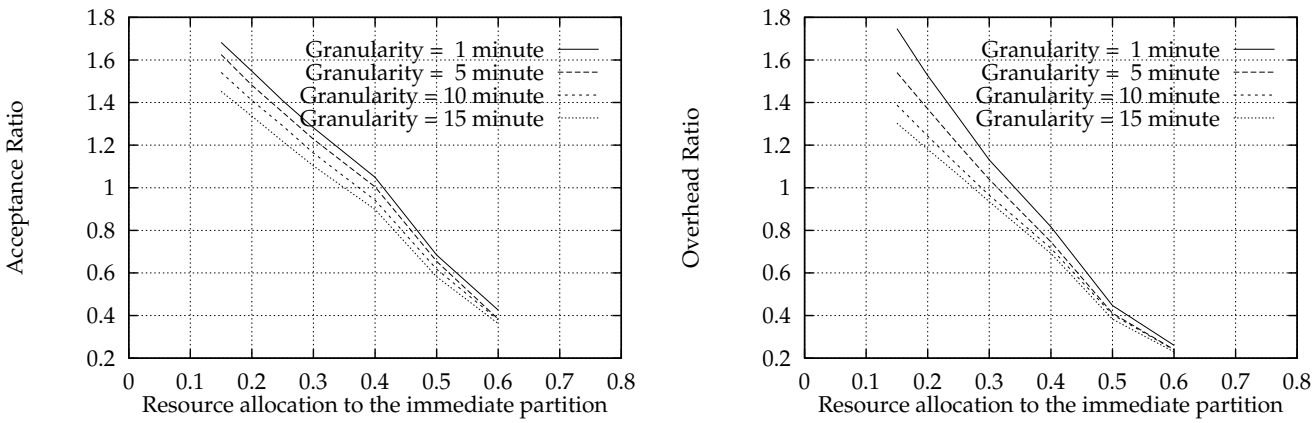

Figure 7: Effect of time granularity

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. Figure 7 shows 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.

5 Discussion and conclusion

We have presented a fully-distributed scheme for advance reservations of real-time connections. The experiments have shown 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 substatial fraction of them, are not established.

We have recently learnt about two other efforts to provide advance reservations for real-time communication. Reinhardt[15] proposes a signaling mechanism for advance reservations with the ST-II protocol and Delgrossi et al.[6] discuss the design issues encountered when trying to support advance reservations in ST-II. The signaling approach in [15] is somewhat similar to ours, except that it uses fixed-size intervals, with the accompanying extra overhead (for example, with five-minute granularity, the overhead for a two-hour video conference would be approximately twenty-four times the overhead for an immediate reservation, which is the extremely unlikely upper bound for overhead in our case). In [5], the authors propose an advance reservation mechanism for *predictive service* [4], which requires characterizing advance reservation

request arrivals, start times, and durations. Since it is designed for predictive service, their approach cannot be used to reserve guaranteed-performance channels.

Acknowledgment

This research was supported by the National Science Foundation and the Defense Advanced Research Projects Agency (DARPA) under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives, by AT&T Bell Laboratories, Digital Equipment Corporation, Hitachi, Ltd., Mitsubishi Electric Research Laboratories, Pacific Bell, and the International Computer Science Institute. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing official policies, either expressed or implied, of the U.S. Government or any of the sponsoring organizations.

References

[1] 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.

[2] 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. *Proceedings of Fifth International Workshop on Network and Operating Systems Support for Distributed Audio and Video*, Boston, MA, April 1994.

[3] 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.

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

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

[6] Luca Delgrossi, Sibylle Schaller, Hartmut Wittig, and Lars Wolf. Issues of reserving resources in advance. *Proceedings of Fifth International Workshop on Net-*

work and Operating Systems Support for Distributed Audio and Video, Boston, MA, April 1995.

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

[8] 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.

[9] 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.

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

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

[12] 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.

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

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

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

[16] 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.