Scaling Issues in the Design and Implementation of the Tenet RCAP2 Signaling Protocol

Wendy Heffner[1]
wendyh@CS.Berkeley.EDU
Tenet Group
University of California at Berkeley &
International Computer Science Institute

TR-95-022
May 1995

Abstract

Scalability is a critical metric when evaluating the design of any distributed system. In this paper we examine Suite 2 of the Tenet Network Protocols, which supports real-time guarantees for multi-party communication over packet switched networks. In particular, we evaluate the scalability of both the system design and the prototype implementation of the signaling protocol, RCAP2. The scalability of the design is analyzed on several levels. It is analyzed with regard to its support for large internetworks, many multi-party connections, and a large number of receivers in a single connection. In addition, the prototype implementation is examined to see where decisions have been made that reduce the scalability of the initial system design. We propose implementation alternatives that are more scalable. Finally, we evaluate the scalability of system design in comparison to those of the ST-II signaling protocol (SCMP) and of RSVP.

Keywords: scaling, multicast connection, multimedia networking, real-time communication, Tenet protocols

1. 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, the Department of Energy (DE-FG03-92ER-25135), 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 author, and should not be interpreted as representing official policies, either expressed or implied, of the
U.S. Government or any of the sponsoring organizations.

1.0 Introduction

In this paper, we take the first step in evaluating of the work of the Tenet Group's Suite 2 project. The Tenet Group's work has been directed toward the issue of providing real-time guaranteed service over packet switched networks [FerVer90]. The Tenet Suite 1 project culminated in the design and implementation of a set of network protocols to provide connection-oriented service based on a simplex unicast real-time channel. In the Suite 2 project, we have extended this work to provide support for multi-party applications. The main goal of our project is to understand the limitations of providing such types of guarantees in this environment. While the group has primarily focused on designing real-time protocols, we believe that evaluation of research solely based upon simulation can sometimes be misleading. An important component of the Suite 2 project, therefore, has been our implementation of the protocol design, as was the case for the Suite 1 project. In evaluating both components, the protocol design and implementation, we must make a strong distinction between choices mandated by the design and decisions made to expedite the implementation. In our evaluation, we have tried to make this distinction clear.

This paper will focus primarily on RCAP2, the signaling protocol of Suite 2, for it is this protocol that must change the most when we made the move to a multi-party framework. RCAP2 coordinates connection setup, teardown, and all other management of connections. In this report, we have chosen scalability as our metric for evaluation of this work. All viable network protocols must be able to support growth. Scaling can be viewed on several levels; for example, as our design was proposed for large internetworks, it must scale to cover large distances and many heterogeneous nodes. In addition, the design must scale to support many multi-party applications of various types. Finally, it must support many heterogeneous participants in a single multi-party application. It is clear that we must consider support for heterogeneity as an important factor when we evaluate the design's ability to scale. In addition, we must address the issue of fault tolerance when we evaluate the scalability of a design based on a connection-oriented approach.

The report is organized as follows: in the next section we provide a general overview of the Tenet Protocols and Suite 2. In Section 3, we discuss the design of the signaling protocol RCAP2, pointing out some of the decisions that affect the scalability of the design. In Section 4, we give an architectural overview and discussion of the prototype implementation of RCAP2. In Section 5, we begin a detailed evaluation of the scalability of the both the design and the implementation, and continue with a discussion of some our proposed changes to the implementation. In Section 6, we discuss some related work, in particular ST-II and RSVP. Finally, in Section 7 we conclude with a summary of the paper.

2.0 The Design of Suite 2

We begin this section with some general background material on the Tenet approach, and then move on to examine each of the Tenet Protocols in particular. This section concludes with a discussion of those components of the system that must undergo change when moving to the multi-party environment of Suite 2.

2.1 Overview of Tenet Protocols

The Tenet Protocols provide mathematically provable guarantees over packet switched networks. The Tenet Protocols take a connection-oriented approach, using resource reservation and admission control to provide these guarantees. Sources specify their traffic, receivers request certain qualities of service, and then admission tests are performed along the path of communication to see if enough resources are available to accept this new channel’s traffic and meet the receiver’s requirements. If the traffic can be supported and requirements can be met, then resources are reserved for the channel. Real-time channels are protected through this process of admission control; when a network becomes saturated, no additional real-time channels are admitted.

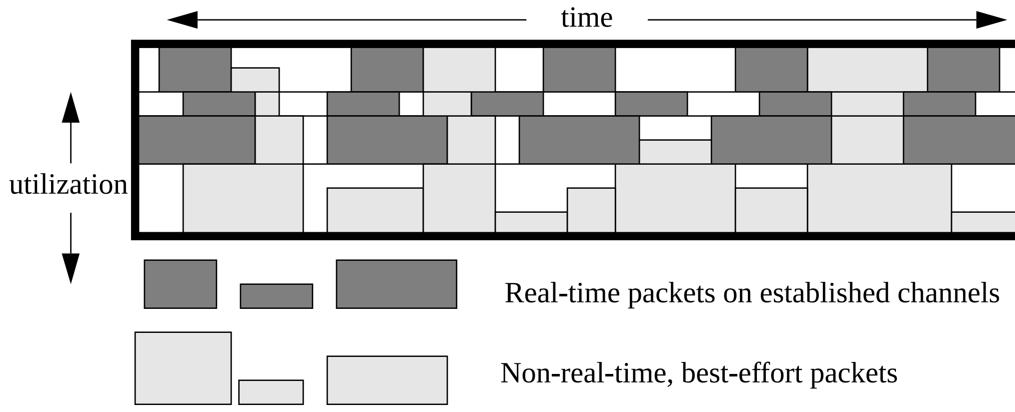
time
utilization
Real-time packets on established channels
Non-real-time, best-effort packets

FIGURE 1: The use of resources allocated to real-time channels.

It should be noted that unused resources “reserved” by a real-time channel may be used by non-real-time traffic. Figure 1 illustrates this usage pattern. The figure shows three established real-time channels with each channel’s resources protected from other real-time traffic. If there is no traffic present on a real-time channel, then best-effort traffic may be scheduled in its place. When present, however, real-time traffic has priority over non-real-time traffic.

Our protocols offer “mathematically provable” guarantees. These guarantees, however, do not restrict our approach to simply providing deterministic *peak-rate* allocation; as shown in [Zhang93], we can provide deterministic guarantees while the sum of the peak rates of all accepted connections exceeds the speed of the link. The Tenet Protocol Suites also support a statistical service, which can co-exist with the more stringent deterministic service. In statistical service, the receiver’s quality of service is specified in terms of delay and loss *probabilities*; thus, the receiver will request a minimum probability that a packet delay will not be greater than a fixed maximum delay and a minimum probability that a packet will not be lost due to buffer overflow. Statistical versions of the admission control tests are then performed to see if these probabilities can be met.

The communication paradigm of Suite 1 of the Tenet Protocols is that of a simplex unicast connection. Channel establishment in Suite 1 is accomplished in a fully distributed way with a single round trip. On the forward pass, admission tests are performed at each node along the route. If the tests at a node are successful, then resources are tentatively reserved for the channel, and the establishment message proceeds to the next node on the path. If anywhere along the route an admission test fails, then the entire establishment fails, and any previously reserved resources are released. Given a successful completion of the forward pass, the aggregates of minimum delay and jitter bounds that are accumulated along the path are compared against the quality of service requested by the receiver. If its requests can be met, then establishment succeeds, resources are committed to the channel, and rate control and scheduling information is passed on to the data delivery network protocol for scheduling and policing the newly established channel. Note that in Suite 1, if the quality of service requested by the receiver has been over-met on the forward pass, then distributed relaxation of the reserved resources will occur on the reverse pass.

In order to address the issues of multi-party applications, Suite 2 of the Tenet Protocols moves from the simplex unicast channel paradigm to one consisting of a 1xN multicast connection. Suite

2 continues to support real-time guarantees through admission control and resource reservation. As we will see in Section 3.1.1, although the establishment process must be modified to cope with several receivers, it still maintains a single round trip approach similar to the one outlined above to keep the setup time to as small a value as possible.

2.2 Tenet Protocol Suites

2.2.1 Protocol Stack

The Tenet Protocol Suites are designed to co-exist in a node with other non-real-time protocols such as TCP and UDP/IP. The Tenet Protocols can only provide guarantees when used in conjunction with a data link layer that can support packet scheduling with deterministic behavior (e.g., synchronous FDDI, ATM, and switched Ethernet[1]). Real-time data delivery is accomplished through two protocols: RMTP, the Real-time Message Transport Protocol and RTIP, the Real-Time Internetwork Protocol. Signaling in the Tenet Protocol Suites is done through RCAP, the Real-time Channel Administration Protocol.

.

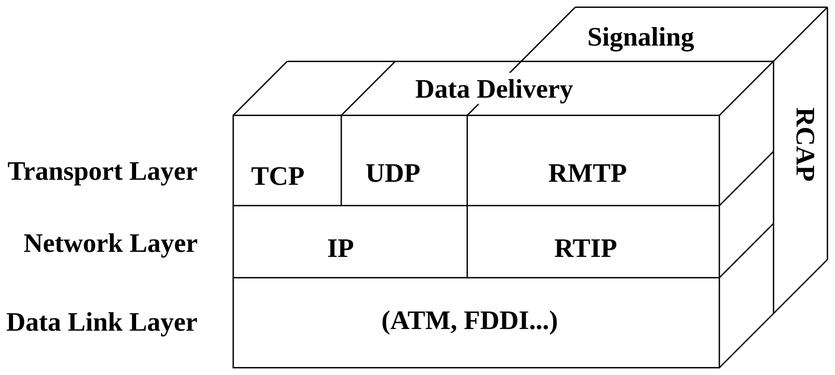

FIGURE 2: Tenet and Internet Protocol Stacks

The Real-time Message Transport Protocol

RMTP is a thin transport protocol that performs fragmentation, reassembly, and source rate control. Because of the real-time nature of our service, we do not implement any ack or nak based schemes of retransmission. We assume that data sent via our protocols is delay-sensitive in nature, and that, if packets are lost or corrupted, they may not be retransmitted within the delay bounds guaranteed to the recipient. It should be noted that, if reliable delivery is crucial, other schemes using redundancy such as forward error correction, linear predictive coding, as outlined in [BolCreGar95], may be more appropriate to traffic with real-time constraints. These redundancy techniques have not been included in our protocol designs.

RMTP does not implement any flow control scheme. Given that the delay and jitter bounds are met by the network, it is assumed that a receiver has made sure at admission test time that it has enough

1. Tenet Protocols cannot provide guarantees on Ethernets that use the standard CSMA/CD collision detection scheme. Exponential backoff creates non-deterministic behavior, which cannot be built upon to provide deterministic or statistical delay bounds to the higher layers.

processing power and buffer space to consume the data sent by the source. In Suite 1, the motivation for this assumption is clear given the single source-destination matching at the time of channel creation. In the context of a unicast connection, it is obvious that a source's data rate would be matched to the specified destination. In Suite 2, there is no direct one-to-one matching between source and destination. Even in this case, however, each receiver will test its ability to consume the traffic the source generates, and will refuse the connection if this condition is not satisfied. Section 5.1.3 discusses this issue in more depth, and details how Suite 2 supports heterogeneous receivers that may require different data rates.

The Real-Time Internetwork Protocol

RTIP, our network layer protocol, performs rate control, jitter control policing, and packet scheduling. RTIP is an unreliable connection-oriented protocol that enforces guaranteed performance. Since channels have fixed routes, packets are guaranteed to be delivered in order, unless nodes permit overtaking within a connection. The connection-oriented approach eliminates any need for per-packet routing, since routing for the connection is performed prior to establishment.

The Real-time Channel Administration Protocol

RCAP is a signaling protocol that performs connection setup and teardown. As discussed in the previous section, connection setup is achieved through admission control and resource reservation. In addition, RCAP supports queries on the state of the channels.

2.2.2 Changes when moving from Suite 1 to Suite 2

Some of the previously mentioned protocols must change more than others when multi-party applications are to be supported. For example, RMTP need not change in the move to multicast. RTIP must be modified to support packet replication on multiple outgoing links at the branching nodes of a connection. In addition, as will be discussed in Section 3.1.6, RTIP must enforce rate control over groups of channels that are participating in resource sharing.

RCAP must change radically in the move from unicast to multicast connections. Channel establishment now must progress across a multicast tree. Futhermore, as outlined in the next section, several additional mechanisms have been added to RCAP2 (the Suite 2 version of RCAP) to support scaling for multi-party communication. The remainder of this paper focuses on the design and implementation of RCAP2, and on how it supports scaling while servicing multi-party applications.

3.0 The Design of RCAP2

RCAP2 is the signaling protocol for Suite 2 of the Tenet Protocols. The protocol provides services for both real-time connection administration (e.g., connection setup and teardown) and information management (e.g., object creation, deletion, and querying of object state). Suite 2 has introduced long-lived objects, so that object creation and deletion is now decoupled from object use. This decoupling allows the client application to reuse objects without incurring the overhead of the object instantiation. For example, if a number of objects are created to describe a distributed weekly meeting, then these objects can be reused from one week to the next, simply by requesting setup and teardown of the connections that they describe.

We have taken pains to use a modular approach when designing our protocols. We have tried to decouple our design from any specific admission control tests and traffic models. This modular approach not only allows for increased support for heterogeneity, but also allows us to experiment with different design choices.

In this section, we will begin by defining some basic terms that will be used in the remainder of the document, and then move on to discuss in detail the mechanisms we added to support multi-party communication.

Channel

In Suite 1 of the Tenet Protocols, a simplex unicast connection, called a *channel*, is an active entity that is in existence solely during the life of the connection. In Suite 2 the term *channel* has different semantics than in Suite 1: in this framework, a *channel* refers to the passive entity that includes all data describing the channel and the current state the channel is in. A channel in this context can be either established (the source is connected to the destination set) or not established.

Target Set

A *target set* is a logical grouping of receivers and the performance requirements specified by them (see below). This grouping allows for decoupling of senders and receivers. For example, a source does not have to keep a list of the individual receivers of its data; rather, a channel's participants are defined simply as its source and its target set. Conversely, if a receiver joins a target set, then connection is automatically attempted to all active channels sending to that target set, dynamically attaching the new receiver without the direct knowledge[2] or involvement of any of the sources. In addition, the target set abstraction enables heterogeneous receivers to specify differing performance requirements.

Sharing Group

A *sharing group* is a set of channels with some known collective pattern of use. When creating this group, the client application has specified to the network what that usage pattern is, thus enabling the network to exploit this information to optimize the allocation of network resources.

Traffic Characteristics

Traffic characteristics consist of a description of the speed and amount of data that a source can generate. Several traffic models have been proposed to describe sources, such as Leaky Bucket, σ–ρ, Xmin-Xave-I. Suite 2 supports multiple traffic models by using a generic RcapTrafficSpec base class to hide the underlying model. This black box is then passed to the admission control tests, each of which can be coded to accept multiple traffic models.

Performance Requirements

Performance requirements are quality of service parameters that describe the way in which each receiver wishes the data to be delivered. These requirements can be specified in terms of desired end-to-end delay, delay jitter, and maximum packet losses due to buffer overflow. In Suite 2, these requirements are presented as ranges of acceptable performance bounds which should eliminate or at least reduce the need for multi-phase renegotiation. The client specifies a desired performance bound and a range that extends to the worst performance bound that he or she might accept. For example, a client may specify its delay bounds requirement as (D_{max}, Δ). Dmax is the desired

2. The sender may request to be notified when additional receivers join. The design of RCAP2 provides mechanisms that support this type of event triggering.

delay bound, however the client will accept delay bounds up to $D_{max} + \Delta$. The ranges are specified by assigning the desired value and the maximum deviation.

3.1 Services of RCAP2

While it is quite possible to support multi-party applications through the use of several Suite 1 unicast connections, this approach would neither be efficient nor scalable. For this reason, in Suite 2 we have moved to a basic 1xN paradigm for a connection.

We modified several aspects of the existing mechanisms for channel establishment to support the move from Suite 1's 1x1 paradigm to a 1xN paradigm in Suite 2. In addition, in moving from unicast to multicast, we added several mechanisms to the protocol suite to improve multi-party communication:

- multicast channel establishment
- dynamic join and leave
- resource partitioning
- advance reservation
- third-party coordination
- resource sharing
- dynamic traffic management

The goals of these mechanisms fall into two broad categories: adding flexibility for the application, and optimizing resource allocation. For example, in order to provide flexible support for the applications, we do not want to restrict a connection to having a fixed permanent set of receivers. This would mandate the policy that all receivers be assembled prior to making the establishment request. In Suite 2, receivers may join or leave an active connection, through a technique called *dynamic join and leave*. This technique allows for attachment of new receivers to, and detachment of existing receivers from, an ongoing connection. In addition, we expect that applications may want to reserve resources in advance in order to guarantee that they will be available at the time of use. Suite 2 supports this type of pre-allocation of resources through a technique called *advance reservation*. Advance reservation, in turn, is supported by the underlying mechanism of *resource partitioning*. Finally, we expect that complex multi-party applications may wish to use a coordinator to create channels, target sets, sharing groups and to arrange for such actions to occur as channel establishment and dynamic join. In Suite 2, we allow for authorized *third-party coordination*, on behalf of the actual participants

The second category of mechanisms support optimization of resources allocated to multi-party applications. These mechanisms can be used by applications to more accurately specify their expected use of network resources. For example, as the number of senders increases in a large multi-party application, it is possible that there will be some known pattern of use that may be exploited to optimize the allocation of network resources; for instance, some floor control techniques used by the application may restrict the number of concurrent senders. We can optimize through *resource sharing* by only allocating enough resources on shared paths to support the maximum number of concurrent senders allowed by the application. In addition, our protocols allow for media scaling through *dynamic traffic management*. Through this mechanism, a source may modify the amount of reserved resources to track long-term fluctuations in its traffic. This technique allows for better overall network utilization by controlling the source's traffic.

The following sub-sections discuss each of these mechanisms in more detail.

3.1.1 Multicast Establishment

Connection setup consists of two phases: a *preparation phase*, which includes routing and an *establishment phase*, which performs admission control and resource reservation. The preparation phase begins by the channel object contacting its associated target set object to obtain the current list of receivers and their requested performance requirements. This information is used to compute a route from the source to the set of receivers. Routing can be accomplished either by a server that has some knowledge about the current network state [Widyono94], or it can be dynamically calculated [BierNonn95]. There are some obvious scaling issues associated with a centralized version of the first approach. These scaling problems can be alleviated by distributing the routing server and by using a routing algorithm based on successive refinements [Moran95]. We discuss this scaling issue in more depth in Section 5.1.1.

The second phase of connection setup is establishment. Although establishment follows the single round trip approach developed in Suite 1 of the Tenet Protocols, we now must scale those techniques to service a (potentially large) number of heterogeneous receivers [BetFerGupHe95]. The multicast establishment process will be presented here as proceeding from the source to the destinations and back. The reader, however, should keep in mind that this process can be implemented in the reverse direction with very few modifications.

On the forward pass of establishment, at each node (router, gateway or switch) on the multicast tree admission control tests are performed using the resources available in the channel's *partition* (see Section 3.1.3). A given set of resources are administered by a *resource server*. For example, at a gateway there may be a CPU server and several outgoing link servers, or in a crossbar there may be simply the outgoing link servers. A link resource server would control the allocation of all resources associated with an outgoing link (e.g., output buffers, bandwidth and scheduling), and it would administer admission control tests for partitions that were represented on that link.

If all tests succeed at a server, resources are tentatively reserved to support the new channel. In addition, if all admission tests succeed on a path through a node, then the calculated minimum local delay bound is accumulated into the aggregate minimum delay bound and is passed in the establishment message on to the next node on that path. If any of the tests fail at a server, then no resources are reserved and admission fails for all destinations on a path through that server. If tests succeed on some servers, then resources are reserved and the establishment message is forwarded only on to links that succeed in *all* their admission control tests.

When the establishment message reaches a node that contains a receiver, a destination test is performed. In this test the accumulated delay passed in the message and the jitter contributed by the last node are compared with the ranges that have been requested by the receiver. If the delay bound and the jitter bound meet the receiver's requirements, then the test succeeds and connection is successfully established to that destination. If this node is a leaf, the reverse pass of establishment is started, otherwise the regular admission tests are performed and the forward pass continues. On the reverse pass, resources are committed on successful paths of the connection, and RTIP2 is given rate control information in each node. At each branch node along the reverse pass route, the establishment process waits for all reverse pass messages from sub-trees reachable from that node. On links that support at least one successful receiver, resources are committed, and rate control information is passed to RTIP2. On links where no destinations can be successfully reached, resources are released, and that link is pruned from the multicast tree. Figure 3 illustrates this process, for a multicast tree with three intended receivers located at leaf nodes of the tree. On the forward pass, an admission control test fails for an outgoing link of a node on the right subtree. Since no other destinations are reachable from that node, the upstream node is informed of the failure. All other nodes and links succeed on their forward pass admission tests, and resources are tentatively reserved. When the establishment messages reach the left subtree's leaf nodes, the requested performance requirements are compared with the achievable bounds. In our example, the requested bound has been met for the middle node but not for the far left node. At this point, the reverse pass begins, and resources are committed on all links that are on the path from the source to the middle node. Both the right path from the first branch node and the left path from the left node of the second set of branch nodes will be pruned from the tree. Resources are released on these links.

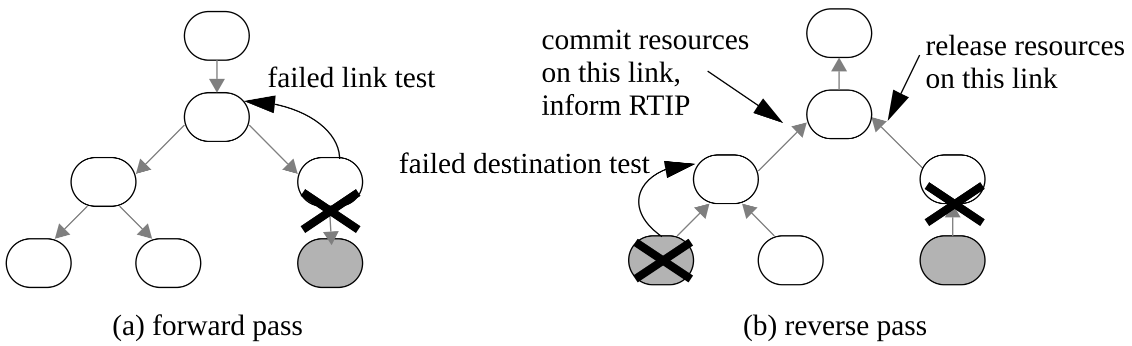
failed link test
(a) forward pass
commit resources on this link, inform RTIP
release resources on this link
failed destination test
(b) reverse pass

FIGURE 3: An example of multicast connection establishment

It should be noted that the establishment process has been fully distributed, and that all local admission control decisions are based upon data that are available at that node. There is no centralized or global decision making. Our connection-oriented approach, however, makes the Tenet Protocols vulnerable to node failure. Members of the Tenet group have addressed this concern in their work on fault handling [ParBan94]. Their approach to this problem uses rapid rerouting for fault recovery. Although their work was done in the context of Suite 1, there is no reason that these mechanisms cannot be extended to the multicast environment addressed by Suite 2. Object state, however, increases the complexity of providing fault tolerance and recovery, for now a failure at a node not directly on the path of a channel may result in the loss of data relating to that channel. It should be noted that this type of failure will not disrupt the delivery of data on the affected connection, but it will simply affect the future management of it. Certain policy decisions such as placing the channel object at the source node's RCAP2 daemon can make our design more robust with respect to this type of failure. In addition, target set information can be dynamically reconstructed by a message traveling along the path of a connection.

A subtle issue is embedded in the above description of establishment. In Suite 2 we allow for what we term *partial establishment*: the failure of a receiver to be connected does not prevent the overall channel from being established. We have chosen these liberal semantics because we feel that they are more flexible. For example, if an application requires an *all-or-nothing* policy, the service provider can build a session layer above our protocols that supports the desired semantics. This session layer can request establishment, and check the results of that attempt before the connection is handed off to the application. If the requirements have not been met, the channel can be torn down, and failure can be reported to the application. If we had chosen to restrict our interface to more limited semantics (i.e., all-or-nothing), then it would be extremely difficult for a service provider to supply an interface to applications that allows for a more liberal policy.

3.1.2 Dynamic Join and Leave

Dynamic join begins with a request to insert a receiver into a target set [EffMul93]. Dynamic attachment is attempted to all channels currently established to the target set. Attachment follows a process similar to the one outlined in the previous section. In the first phase, a route is found from the receiver to the nearest attachment point on the multicast tree. In the second phase, establishment occurs in a single round trip. The forward pass progresses from the receiver through the nodes leading to the attachment point. On each node along the route, resource server admission control tests are performed with respect to the channel's partition for the reverse direction. If all tests on the forward pass are successful then the aggregate delay bound accumulated on the forward pass is added to the delay bound from the source kept at the attachment node, and the total is compared to the bound specified by the joining receiver. If this test is successful, then resources are committed on the reverse pass, and RTIP2 is given the rate control and scheduling information in each node.

Although it is quite possible to implement initial establishment in the reverse direction, we can still achieve "receiver-oriented" establishment without changing the implementations as described above. Even if initial establishment progresses from the source to the receivers and back, we can still have receiver-initiated establishment by leveraging off of both partial establishment and the join mechanism. A channel is first established to an empty target set and then receivers can join. Although it is quite possible that a non-optimal multicast tree might result from building the tree through this type of connection establishment, it is also unclear what impact several dynamic joins and leaves would have on a multicast tree developed through a source-initiated establishment.

Dynamic leave results in a receiver being removed from a target set and from all channels sending to that target set. All links in the multicast tree exclusively servicing that receiver are torn down by releasing resources allocated to that channel in RCAP2 and requesting RTIP2 to remove the rate control and scheduling information for that channel on that link. It must be noted that, if the node at which the receiver is located also serves other destinations then no resources are released.

3.1.3 Resource Partitioning

Resource servers control a set of resources either for the CPU, a switch, or an outgoing link. A resource partition (or simply *partition*) is a logical subset of resources (e.g., percentages of buffer space, bandwidth or computation power, and schedulability) available on a resource server [FerGup93]. Each resource server, therefore, may have its resources divided amongst one or more resource partitions. See Figure 4 for an illustration of a node with two partitions and four resource servers (i.e., one CPU server and three link servers).

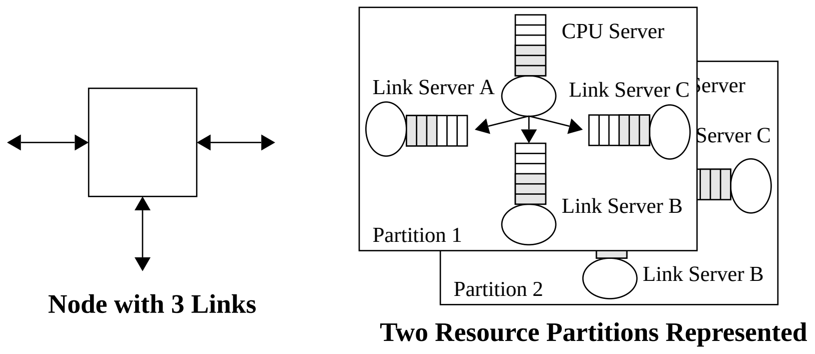

FIGURE 4: Illustration of Resource Partitioning

When a channel is created, it must specify the partition it wants to belong to. During establishment, admission tests for that channel are performed on the resources available in the channel's specified partition. If there are not enough resources in the specified partition to support the new channel, then it will fail that admission test. Channel establishment decisions, therefore, must only take into consideration the other channels established in that one partition. Resource partitioning is a logical division of resources that is only needed during admission control, and thus is only known to RCAP2, the signaling protocol. Once a channel is successfully established, the concept of resource partitioning is not needed for rate control enforcement in the data delivery protocols.

Network administrators can use resource partitioning to enforce several different policies with regard to resource usage. For example, partitioning allows the network to be divided into several virtual private networks. It can be used in this fashion to restrict specific traffic from or to a particular sub-domain or set of nodes. In addition, resource partitioning can be employed to enforce a hard boundary protecting resources for non-real-time traffic, since it allows a network administrator to restrict real-time traffic to use only some subset of the available resources at a node. Recall that non-real-time traffic may still use the unused resources of any partition, while real-time channels may only use resources in their specified partition.

3.1.4 Advance Reservation

Advance reservation allows channels to be set up for a fixed interval in the future [FerGupVen95]. This special case of establishment allows distributed applications to more easily coordinate a large complex communication structure by reserving the resources for it in advance of their use. This is analogous to booking hotel rooms in advance for a conference. Presumably, if there is denial of service, then the application has some time to adjust and make alternative plans.

Resource partitioning supplies the mechanism that provides the underlying support for advance reservation. *Advance channels* are segregated from channels that request immediate establishment (termed *immediate channels*) by assigning resources to them on a separate partition. This allows the network administrator to separate out a group of resources, and assign them in the future to channels that are established in advance. These resources are protected from use by immediate real-time traffic. Suite 2 requires that advance channels not only declare their start time but also their duration. By having a declared start and end time, advance channels can be assigned to a fixed schedule, and resources can be immediately reassigned to other channels upon the scheduled termination time of a channel. In Suite 2, immediate channels do not specify a duration; hence, resources are assigned to these channels until they are explicitly released by the application.

3.1.5 Third Party Coordination

Suite 2 supports location independence through *third party coordination*. Objects can be created, and actions can be performed upon those objects (e.g., establishment, join, and queries), by authorized third party agents. This mechanism supports the concept of network resource booking agents. When used in conjunction with advance reservation, it further supports this natural model by allowing a conference organizer to set up in advance complex communication patterns on behalf of the participants.

Channels, target sets, and sharing groups may be created by any *authorized* requestor. In our preliminary design for authorization, we took a capability-based approach where the object creator has rights to create, destroy, and request actions on, an object (e.g., request the establishment of a channel). The creator can grant subsets of these rights to others. To request an action on an object, the requestor must pass a *handle* to the object. This handle contains the requestor's capabilities for that object. RCAP2 checks the requestor's authorization for the action before allowing the request to proceed.

Although both advance reservation and third party coordination are techniques that primarily support flexibility, they also simplify the process of creating complex multi-party connections. A single organizer can coordinate all object creation and connection setup, and thus simplify the organization of conferences that might otherwise reach levels of complexity that would be difficult to administer.

3.1.6 Resource Sharing

We directly address the issue of scalability by allowing for resource sharing. Through this technique, an application can specify usage patterns over a group of *related* channels, and the network can exploit this knowledge by optimizing resource allocation [GuHoMoNg95]. Related channels may share resources, and thus exploit statistical muliplexing gains. Although rate control is performed over the group of related channels, the network still guarantees that the requested performance bounds will be met over the individual channels.

To exploit resource sharing a client application creates a *sharing grou*p with a *sharing specification*. This sharing specification contains an aggregate traffic specification and a *threshold* number. Channels are then added to the sharing group. Resource sharing is implemented on a per-resource-server basis. As each channel in the group is established, it is added to the number of related channels previously established at that server and compared against the sharing specification threshold. Prior to reaching the threshold, channels are established normally, as detailed in Section 3.1.1. Once the threshold is exceeded by the new request, the aggregate traffic specification is used for admission tests. If the tests are successful, then resources are reserved for the aggregate traffic specification. In addition, the resources allocated to the channels previously established in the group are released[3]. Once the threshold channel has been established, subsequent related channels need not perform admission tests on those links.

Resource sharing increases network utilization by allowing related channels to exploit statistical muliplexing gains. In addition, for most applications we can expect that there will be some maximum concurrence that remains fixed while the number of participating channels may increase.

3. We should note that these resources are available during the performance of admission tests using the aggregate traffic specification.

This pattern of use suggests that this is a very scalable service; very few additional network resources may be needed as the number of participants continues to grow.

3.1.7 Dynamic Traffic Management

Traditional admission control provides for a one time negotiation for network resources. A source is expected to supply a bounding traffic specification, and the network performs tests to see if it can meet the desired service requested by the destinations. Through dynamic traffic management (DTM), we allow the sender to renegotiate this contract during the life of its connection. Senders can now bound their traffic on a smaller time interval than the life of a connection.

It is obvious that deterministic service best serves constant bit rate (CBR) traffic. To obtain high utilization the application must send at a rate as close as possible to its maximum rate. Generating an accurate least upper bound is trivial for CBR traffic. Unfortunately, several multimedia applications employ compression, and thus generate variable bit-rate (VBR) traffic. Recent work has addressed the issue of incrementally bounding VBR traffic [ZhaKni95b][GroKesTse95], thereby getting a more accurate bound for each time interval. To reap the benefits of this work, we must assume that the network supplies a mechanism to allow for renegotiation of the resources allocated to a connection. Dynamic traffic management is such a mechanism.

In DTM[4] we use a two-round-trip technique for channel renegotiation, similar to the two-phase commit used in distributed databases. Our semantics require that all current receivers of the channel must be supported after the renegotiation. This requirement forces us to use a two-round-trip approach: the first round-trip tests to see if the new traffic requirements can be met by *all* links and nodes servicing the current destination set, and the second round-trip commits or aborts the DTM event. Whether or not all destinations may be successfully supported with the new traffic specification is only known at the end of the first round-trip. Note that it is possible that a channel may successfully renegotiate through DTM to lower the amount of assigned resources, and later fail in its request to re-obtain the original allocation.

4.0 The Implementation of RCAP2

4.1 General Architecture

RCAP2 consists of two components, a user-level daemon and a library that is linked in with client applications. The RCAP2 daemon services requests from both applications and other RCAP2 daemons. The RCAP2 library provides a set of messages (RcapMessages) and communication mechanisms for sending requests and delivering replies to and from the RCAP2 daemon. RcapMessages are used not only to communicate between library and daemon, but also serve as the base signaling message used between peer RCAP2 daemons.

4.1.1 RCAP2 daemon

The RCAP2 daemon is a user-level process that serves requests from client processes or peer RCAP2 daemon processes located at remote nodes. Requests can access and manipulate resources located at that node. We have designed the daemon with an object oriented structure. Channels,

4. The DTM work is currently in the design stage. The following description of the design may change in some aspects as the work matures.

target sets and sharing groups are realized as derived classes that inherit from a common parent RcapObject class. By using an object oriented approach, we are able to exploit inheritance to provide a common interface to all objects. The basic structure of the daemon consists of an RCAP2 level dispatcher (RLD) that dispatches both *inter* and *intra* RCAP2 messages, a location and naming service (LNS), a series of RcapObjects and RcapObjectManagers, and an establishment subsystem.

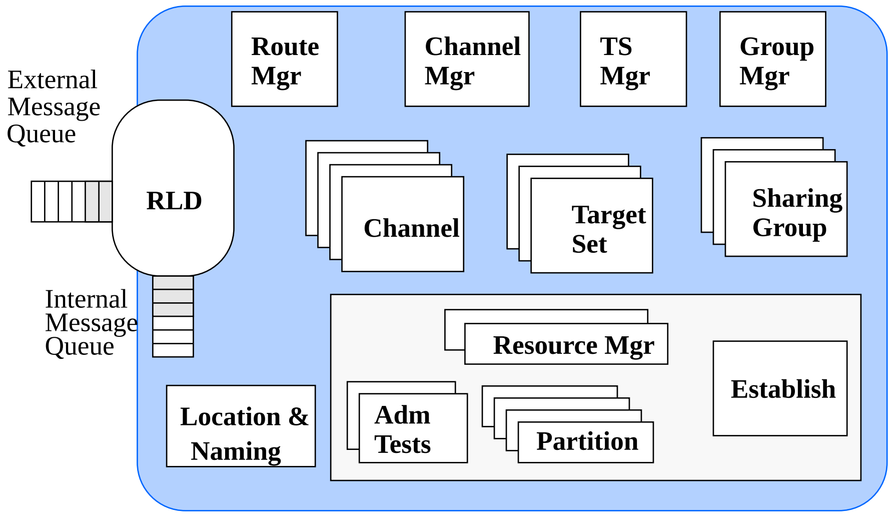

FIGURE 5: RCAP2 daemon architecture

Rcap Level Dispatcher

The RLD services messages sent from outside the daemon (from peer RCAP daemons and from clients) and messages sent from objects inside the daemon (intra-RCAP2 messages). The dispatcher requests a lookup from the LNS on the message destination. The LNS returns a pointer to the destination object if it is located at the current daemon or a pointer to the object manager if the message must be forwarded to another daemon. The dispatcher then passes the message to the object level dispatcher contained within the object referenced by the returned pointer. That object then services the message request or forwards it onto the proper RCAP2 daemon.

Location and Naming Service

All naming and location services reside in this object. By isolating our naming conventions to this object, we can easily modify them as the system grows. In the current implementation, objects are named with an *RcapGlobId*. We have hidden a location based naming scheme within this global ID by concatenating the IP address of the object's location with the object type and number. The location and naming service (LNS) within the daemon performs this object lookup. With the current naming scheme, this lookup is greatly simplified.

Object Managers

Object managers service requests to create, destroy, or forward requests to objects located at other peer daemons. There is a single instance of each type of object manager at a daemon: channel object manager, target set object manager, sharing group object manager, routing manager. The latter manager serves a slightly different purpose: it simply provides a level of indirection so that an object sending a routing request need not know the exact locations of the routing servers.

Target Set Object

The target set object keeps a list of current members and of their performance requirements. In addition, it keeps track of the list of channels that are established to it. As currently implemented, for each target set there is a single representation of the target set object. This design choice has an obvious impact on scaling which we will discuss in the next section.

Channel Object

The channel object is responsible for maintaining global state on a given channel. It is also responsible for initiating the various types of establishment processes (full establishment, attachment initiated by a new receiver joining the channel's target set, full teardown, detachment initiated by a member leaving the target set, and DTM renegotiation). In addition, the channel object is responsible for serializing conflicting establishment events.

Sharing Group Object

The sharing group object maintains a list of the channels grouped for resource sharing and the sharing specification attached to them. As each channel is inserted into the sharing group, the associated channel object is passed the sharing specification.

Establishment Object

The distributed process of establishment is coordinated at each local node by the establishment object. This object requests that admission tests be performed on a given partition's resources at each server involved in the connection (see Figure 4). Our design allows for different admission control tests to be used on different servers. For example, one server may use rate control static priority (RCSP) tests and another may use earliest deadline due (EDD) tests. This is acceptable as long as a single scheduling discipline is used consistently for admission control testing in a server. On the reverse pass, the establishment object releases resources on servers leading only to receivers that cannot be reached. In servers leading to one or more receivers that can successfully be reached, RTIP2 is given scheduling and rate control information.

4.1.2 RCAP2 library

The RCAP2 library consists of a set of RcapMessages that can be used to call a variety of RCAP2 object methods. The low-level client interface to RCAP2 is based on asynchronous, message-based, request-reply communication. This interface includes requests to:

- create objects: ChannelNew, TargetNew, SharingNew;
- destroy objects: ChannelDelete, TargetDelete, SharingDelete;
- trigger establish events: Join, Leave, Establish, Teardown, DTM;
- query the state of objects and connections.

The RCAP2 library contains methods to open and close a connection to the local RCAP2 daemon, and to dispatch messages from and to the client and daemon.

4.1.3 RcapMessages

An RcapMessage is a versatile object oriented data structure used in both *inter* and *intra*-RCAP2 communication. RCAP2 uses RcapMessages for communication between an application linked with the RcapLibrary and the local RCAP2 daemon. In addition, RcapMessages are used to communicate between peer RCAP2 daemons. Messages with different payloads are derived from a single RcapMessage base class; thus, they share a common interface.

When sent as an intra-RCAP2 message, the structure remains in an *unpacked* state. In this state, the fields of the message payload are directly accessible, and the message header is protected through access methods supplied by the class. If it is sent for inter-RCAP communication, the RcapMessage is first *packed*: the fields of the RcapMessage are first converted into network byte order and then packed into a character array member field of the class. The `send()` method of the message takes care of both packing and reliably sending the message. To receive a message, the header is first read, and then it is cast as the proper derived class; finally, the remaining payload is read and unpacked into the fields of the message.

4.2 Implementation Decisions

We made a number of simplifying decisions while constructing the prototype implementation. These decisions have ramifications for the scalability of the prototype. In this section, we detail some of these decisions and provide some motivation for each choice.

1. RCAP2 library only contacts the local RCAP2 daemon

   In the prototype we assume that communication between the client application and the RCAP2 daemon must occur between processes located on the same machine. This decision requires that there be an RCAP2 daemon resident at any node at which clients reside.

   It is clear that an instance of RCAP2 is required when setting up channels at a node; thus, an instance of RCAP2 is needed at any point where RTIP2 exists. In Suite 1 all requests are generated from either the source or the destination of a channel. In that framework, mandating local communication between the client and RCAP is obvious. If the client is requesting a real-time channel, serviced via RTIP, then there must be an instance of RCAP resident at the node. This need is less clear in Suite 2, where we allow for third party coordination. In this context, clients may request actions that have no effect on the state of their local node.

   We chose this arrangement for two reasons. First, it eases authentication issues; there is a single entry point for a given client. Second, it simplifies the communication pattern to RCAP2 thereby reducing the number of error conditions.

2. no distributed objects

   Each instance of an object has a single representation (e.g., each target set is represented by a single instance of a target set object). There are no distributed representations of any of these objects. Note that this decision does not mandate that all instances be in some central location (see #4 below).

   We chose this policy purely for simplicity of implementation.

3. co-locate channel objects with their target set objects

   We have chosen the policy of co-locating at the same daemon channels objects with their associated target set objects. Thus, when a channel is created, the object representing it is placed within the same daemon at which the specified target set object is located. The normal progression of events begins with the creation of a target set. This is either followed by receivers joining that target set or by the creation of channels sending to it. Each of these events impacts the amount of data that is stored at the daemon at which the target set object is located.

While this policy optimizes the communication between a channel object and its associated target set object during channel establishment, teardown, join and leave, we will see in Section 5.2.2 that it limits the scalability of the design.

4. locate a target set object at node where initial create request was made

   The object representing a target set is placed at the daemon that receives the creation request. Again, this policy was chosen for simplicity of implementation, for it does not require any additional mechanisms to be employed when locating an object.

5. source-initiated establishment and join

   We have implemented establishment starting from the source node of a channel, proceeding down the multicast tree to the receivers, and returning on the reverse pass to the source. Join is implemented in the same manner. The establishment to a new destination starts at the source node and progresses to the attachment point, incrementing reference counts at each node for the number of destinations reachable from that node. When the message reaches the attachment point of the tree, admission control tests are performed on that node and on each hop to the new receiver. If the establishment is successful, then RTIP2 is informed on the reverse path back to the attachment point.

   This join technique has allowed us to leverage off of existing establishment code. In addition, with this implementation, it becomes trivial to add support for bundling join attempts and connecting them *en masse*. This, however, does not permit receiver-initiated establishment, which we believe is desirable or required in a number of applications. Thus, a future version of the implementation ought to use the receiver-initiated attachment outlined in Section 3.1.2.

6. no caching of signaling message connections

   Signaling messages (RcapMessages) are transported reliably via TCP/IP. The `send()` method of the RcapMessage opens a connection to the destination address, sends the message, and then closes the connection. No connections are kept open and cached in our prototype.

   Again we chose this implementation because it was easy to implement. There is far less state information to be maintained. We have found, however, that we incur a large performance hit by opening and closing connections for each message.

5.0 Evaluation of RCAP2

In this section we evaluate the scalability of both the design and the prototype implementation of RCAP2. It is important to draw a clear distinction between the design and the implementation of the protocol. The former can produce many variants of the latter. While we have attempted to design a scalable protocol, we are cognizant that many of our initial implementation decisions have been made at the expense of scalability. What we hope to show in this document is that the design supports scalability, and that it can generate a scalable implementation. This section begins with an evaluation of the design, then moves on to a discussion of the current implementation, and concludes with a sketch of a more scalable implementation.

5.1 Design Scalability

5.1.1 Support for Large Internetworks

To support deployment on wide area networks, our design must scale to large distances with many heterogeneous hosts, routers, switches, and links. Our design must be portable to many environments. Several design decisions have been made with this in mind.

Like Suite1, Suite 2 supports multiple scheduling algorithms. The design of RCAP2 allows us to plug in several different admission control tests depending on the scheduling algorithm being supported by the node. It has been noted [Ferrari92] that, as long as the scheduling algorithms used along the path of a connection are chosen from the very large class of service disciplines that can support real-time communication, then end-to-end guarantees can be met. By not fixing our approach to a single scheduling algorithm such as Weighted Fair Queueing [ParGal93], which requires homogeneity among the nodes, we can support heterogeneous nodes and create real-time channels traversing many heterogeneous networks.

RCAP2 chooses a scalable approach to admission control. Connection establishment decisions are distributed along the paths of the multicast tree. In addition, by reusing the approach taken in Suite 1, the establishment process maintains the low overhead of a single round trip.

The Tenet Protocols are restricted to using data link layers that can support our real-time guarantees. In order for the Tenet Protocols to provide guaranteed service, the underlying data link layer must supply some form of priority scheduling, as well as allow for bounding packet-delivery time. For example, traditional CSMA/CD exponential backoff used in broadcast Ethernets results in non-deterministic behavior. Schemes modifying CSMA/CD by requiring reservation of transmission slots prior to transmission have been shown to support real-time guarantees over such types of LANs [YavPaiFin94].

Tenet’s connection-oriented approach requires that RTIP be present in all routers and switches on the path from source to destination(s). It has been proposed that tunneling (i.e. encapsulation) be used to cross sub-networks that do not support our protocols. If the end-to-end delay and jitter on the tunneled section of the path can be bounded then we can still support guaranteed service, otherwise we cannot [Ferrari92].

Finally the Tenet Protocols assume that routes be pre-computed prior to establishment. This requirement has obvious scaling limitations if one assumes that it requires centralized route calculation. Fortunately, this is not required. Routes can be generated dynamically by distributed discovery or they can be calculated by some distributed form of routing server. The Tenet approach simply requires that they be computed prior to establishment.

5.1.2 Support for Many Multi-Party Connections.

The Suite 2 protocols must gracefully support many multi-party connections. This requires that the protocols make efficient use of network resources. Strict peak rate allocation for bursty VBR traffic traditionally does not provide high network utilization.

Suite 2 supplies several solutions to this problem. First, as stated in Section 2.1, non-real-time traffic may use unused resources allocated to real-time channels. During peak periods for non-real-time traffic, the network may have extremely high utilization. The network utilization will only suffer during periods where there is little non-real-time traffic and several bursty real-time channels with deterministic guarantees. Suite 2 has supplied two mechanisms that address this situation. The first technique is resource sharing among related channels. This mechanism allows for statistical multiplexing of real-time traffic. While it is the onus of the client application to specify the sharing relationship among related channels, one would expect that a tariff structure would reward the client for cooperating with the network in this optimization.

The second mechanism that supports higher network utilization is dynamic traffic management. Through DTM, the source's traffic can be accurately bounded over smaller intervals, thus allowing the peak and average rates to be closer to one another [ZhaKni95a]. As the peak and average rates merge, VBR traffic more closely resembles CBR traffic, and therefore, the utilization induced by a real-time channel increases.

Finally, the protocols support not only deterministic but also statistical services. Statistical service guarantees bounded loss probabilities, and bounded delay probabilities, and allows for more channels to be supported by the same resources.

5.1.3 Support for a Large Number of Receivers in a Single Connection

To support a single large connection, our protocols must scale to support many heterogeneous receivers. These receivers may have different hardware, different amounts of buffer space and therefore, different rates of consumption. In addition, the receivers may be in widely different locations relative to the sender.

In such a scenario, we cannot support prolonged multi-phase negotiation between the network and each receiver. Suite 2 addresses this issue by requiring the receiver to state a *range* of acceptable performance. This range contains both desired bounds and minimum performance. On channel establishment, the network performs a single check to see if the achievable bounds are better than or fall within the ranges requested by each receiver. If it does, then the receiver is admitted; if it does not, then the receiver is not connected. The rejected receiver has the option of leaving the target set and rejoining it with more lax performance requirements.

.

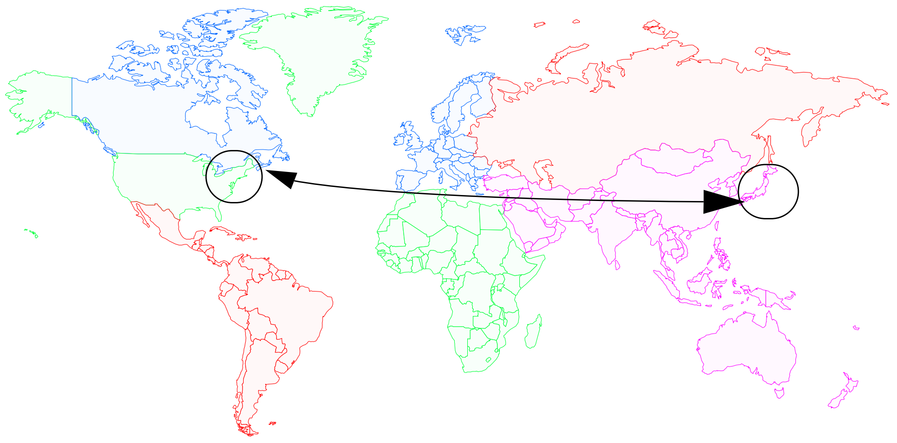

FIGURE 6: Example of geographically disperse receivers

Geographically dispersed receivers may supply performance requirements that reflect their distances from the senders. As shown in Figure 6, a distributed conference may have to take place between several participants located in New York and a single participant located in Tokyo. The participant in Tokyo, in her function as a receiver, must join the main audio and video target sets with must less stringent delay requirements compared to her fellow conferees. If the participant in Tokyo wishes to send both audio and video data to the other participants, then there may need to be additional target sets to reflect the delay in sending data from Tokyo to New York.

Figure 7 shows the two video target sets used to support this conference, and each receiver's specified delay bound component of the performance requirements. In this illustration, Target Set #1 services video that originates in New York, so both participants A and B will send to this target set. Note that both A and B not only send to this target set but receive from it as well. Target Set #2 services video that originates in Tokyo, thus participants A and B must specify longer delays.

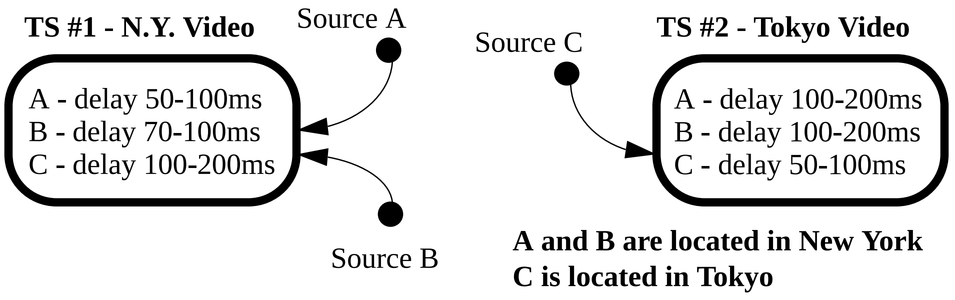

FIGURE 7: Example delay requirements

In addition to supporting receivers requiring differing delays, we must also support receivers with differing capacities. Differences in amounts of buffer space, processor speed and other machine characteristics (e.g., presence or absence of decompression hardware) impact the data rate that the receiver can consume. In addition, low capacity links in the network can prevent reception of high-bandwidth data. Suite 2 can support these heterogeneous receivers through the use of data layers and concentric target sets. Senders hierarchically encode their data sending the base layer to one target set and each enhanced layer to other target sets. Receivers are expected to join multiple target sets depending on their ability to support data rates. All receivers would join a target set receiving the base layer, and progressively smaller subsets of the receivers would join target sets receiving the enhanced layers. Figure 8 illustrates this usage pattern with an example. Receivers A-F wish to receive data at varying data rates: receivers A and B can receive all three layers, receiver C can support both the lowest rate encoded in the base layer and a medium enhanced layer, and receivers D, E and F can only consume the minimum data rate. Three target sets must be created to service this group of heterogeneous receivers. All receivers should join Target Set #1 and receive the base encoded layer. Receivers A-C should join Target Set #2 and receive the additional medium-resolution enhanced layer. Finally, receivers A and B should join Target Set #3 to receive the highest-resolution layer.

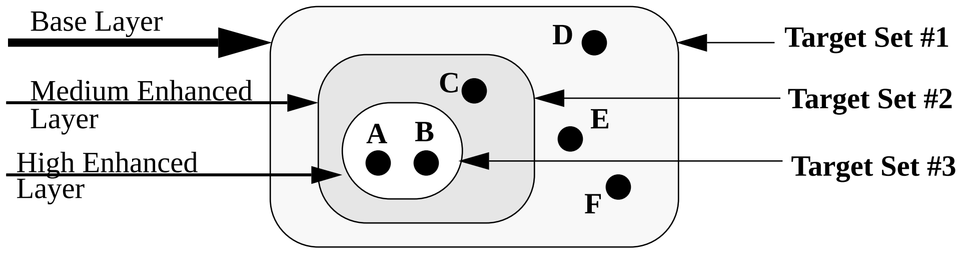

FIGURE 8: Support for multiple data rates

In addition, this technique can be used to circumvent the problem of a bottleneck link on the path of a connection. As illustrated in Figure 9, by layering the code and establishing the channels in the correct order, we can detect the highest data rate that the network can support within the range of layers that the receiver has joined. Continuing the example above, we must begin by establishing the base-layer channel, then move on to the medium-enhanced layer, and conclude with establishing the high-resolution layer channel. If there exists a bottleneck link on the path that exclusively serves receiver A, then one of two scenarios can occur: either the base-layer channel succeeds in its establishment (and possibly other layer channels), or the base-layer channel fails. If the former situation occurs, then receiver A at least can create a lower resolution data stream out of the components it receives. Through this technique, A can determine a rate close to the maximum one its path can support and, if the rate is acceptable, participate in the application anyway.

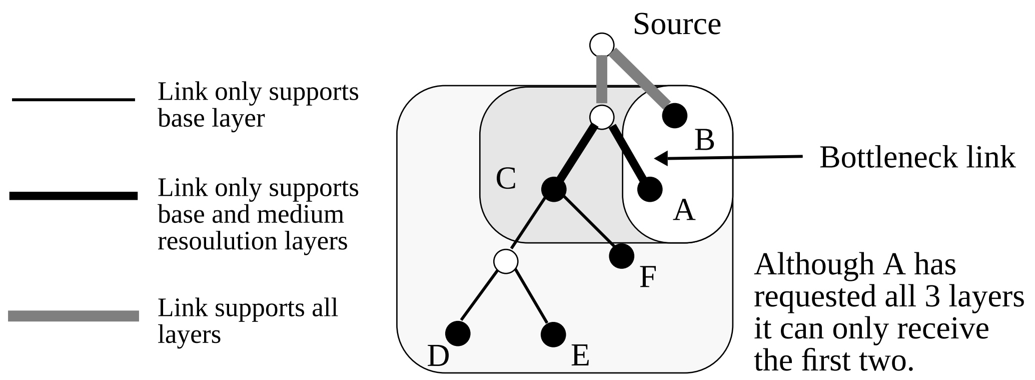
Source
Link only supports base layer
Link only supports base and medium resoulution layers
Link supports all layers
B
Bottleneck link
C
A
F
D
E
Although A has requested all 3 layers it can only receive the first two.

FIGURE 9: Using hierarchical encoding to support receivers serviced by low bandwidth links.

5.2 Implementation Scalability

Several of the implementation decisions discussed in Section 4.2 impact scalability. Many of these decisions were made to expedite the implementation effort. A more robust implementation will need to incorporate changes such as those detailed in the next section. This section of the document flags the non-scalable implementation decisions.

5.2.1 No distributed objects

The prototype implementation has only a single representation of any instance of an object. As the number of participants in a multi-party application grows, this approach not only creates a classic hot spot of activity, but also can lead to the object's size growing beyond the capacity of the daemon process holding the object. For example, as each receiver joins a target set, the size of the object representing that target set increases. In addition, more state information is maintained by the target set object as the number of channels established to it grows.

Both a target set object and a channel object can be a hot spot in large multi-party applications. It is clear from the discussion above that the target set object must be actively involved in joins, leaves, establishments, and teardowns. Since connection setup is coordinated through a channel object, this object can be a bottleneck as well when many receivers join and leave the channel's associated target set. The channel object is a serialization point for establishment events[5]. This serialization is needed to ensure complete matching between senders and receivers during such times as concurrent channel establishments and receiver joins. While processing one establishment request, other establishment requests are enqueued until the processing of the pending event finishes. When that processing is completed, our implementation has taken the scalable approach of bundling queued join requests and connecting them in a single attachment process.

5.2.2 Co-locating channel objects with their target set objects

While reducing communication overhead, the decision to co-locate channel objects with their target set objects may detract from the scalability of the design. As the number of senders to a target set increases, the data representing those channels will be located at the same node, and all activity for that application will concentrate on that single daemon.

5.2.3 Locating the target set at the node that received the creation request

At first glance, the simple policy of locating the target set at the node that received the creation request seems to be scalable. In Suite 2, however, we wish to support the possibility of a third party coordination. In a sample scenario, a third party coordinator may create several target sets, channels, sharing groups, join many receivers, and then request advance reservation for the channels. With our current policy, *all* objects created by that coordinator would be placed at that one node. When we scale this example to the case in which the coordinator creates several large conferences, we may overrun the resources and abilities of that RCAP2 daemon.

5.3 Scalable Implementation Proposal

It is clear from the previous section that we must distribute some of the objects in RCAP2. We begin with a discussion of how each object scales, leading us to understand which objects must be distributed and when. We then detail how various actions such as object component creation, object component lookup, establishment, join and DTM might occur in this framework. Finally, we conclude with an analysis of this proposal and of a possible alternative to it.

5.3.1 Object Scaling

We begin our discussion about distributing object state information by examining *what* objects present scaling problems and *when* do they present those problems. Our analysis covers both object size and object activity.

Target Set Object

The target set object presents a scaling problem in scenarios where a large number of participants are using a single target set. Recall that the target set object is an object that maintains state information on current members and currently established channels sending to it. As either the number of members increases or the number of *active* (i.e., established) channels increases the size of the target set object grows. A more critical scaling issue may be the one of activity. This object must be contacted during most establishment events (with the exception of DTM). In addition, it must coordinate all requests by receivers to join and leave.

5. Establishment events include initial establishment, incremental attachment or detachment triggered by a join or leave, DTM scaling, and full teardown.

Channel Object

The channel object size grows only when it is actively involved in an establishment event. During this process, the object enqueues incoming establishment requests. As each request is serviced, it is dequeued and is no longer kept. Therefore, unlike the target set object, the amount of data maintained by a channel is not monotonically increasing with the number of participants. The amount of cached data simply increases at times of concurrent establishment activity. Current connection state information is not kept in the channel object; it is maintained in a distributed fashion in a separate data structure along the nodes of the connection. The channel object does not keep a list of the currently connected receivers. That information may be discovered by sending a message along the path of the channel.

It should be noted that a *passive* (i.e. not established) channel object has a fixed size, and its associated target set is unaware of its existence. Joins to the associated target set do not impact a passive channel object; thus, a channel object in this state cannot be considered a hot spot.

Sharing Group Object

The sharing group object simply contains a list of channels in the sharing group, and the sharing specification attached to the group. It cannot be viewed as a hot spot, for it is only contacted upon channel creation, and is not on the critical path of establishment. The amount of data maintained by the object increases with the number of channels participating in resource sharing, regardless of whether or not the channels are established. This object can be viewed as a scaling problem only if we believe that this state information might grow past the capacity of the daemon in which it resides. Unlike the channel object, we have not instituted any sharing group object placement policy that would require co-location with any other objects. Scaling problems, therefore, are not compounded by any location policies.

5.3.2 Distribution of Object Components

It is clear that we can make our implementation more scalable by distributing the target set object. This can be achieved by either placing the components regionally (i.e., one per domain), or by a more flexible policy that triggers component creation when a target set's membership reaches some high watermark. Each policy has its own distinct advantages. While the latter policy is more flexible and allows for better scaling, the regional placement policy more naturally meshes with our approach to distributed routing. For purposes of discussion we will assume that a regional placement policy will be adopted.

Each target set object component contains data about the members located in a given domain and a list of the locations of the other components of the target set object. At the time of creation, a single component of the target set object is instantiated. The generation of additional target set object components is triggered when non-local receivers are added or attempt to join. A similar technique for distributing objects has been used in such systems as the Grapevine mail service [SchBirNeed84]. In Grapevine, each GV Registry must know about all other components of the GV Registry. Experience with the Grapevine system has shown that there are some scaling limitations to this design. Xerox PARC addressed these problems in a later version of the mail service, Clearinghouse, by adding another level of indirection and creating a hierarchial system of registries [OppDal83]. The limitation seen in Grapevine may not apply directly to our use of the technique. In Grapevine, the GV Registry system must cover all areas where mail is delivered. In our system, a target set object must only cover regions where there are participants in that particular target set. Thus, the regional scope of the problem is in general quite different.

The distribution of channel object components raises some interesting questions. As we have seen in the previous section, the passive channel object itself does not need to scale. However, after establishment, the channel object becomes active, and a potential hot spot for future joins, leaves, and for DTM events. If a channel is sending to a very large target set, the initial establishment may also suffer from a scaling problem. Recall that a channel object begins establishment by requesting the current membership of the target set. The channel object then requests a route from the source to this list of participants. Given this procedure, we should also attempt to distribute the coordination of initial establishment so that each component of the channel object may independently request routing for some subset of the participants. In other words, if a single object must coordinate a large list of participants during an initial establishment, then we still have a scaling problem.

5.3.3 Distributed Actions

Component Creation

As previously stated, target set object component creation is triggered when a non-local receiver is added to or attempts to join any of the components of the target set. Channel object component creation is triggered during establishment. A component of the channel object is to be placed in each region in which a component of the channel's target set object resides. To make the system more scalable, the local components of the channel objects should not be co-located with the local target set object components. Since we have concluded that the sharing group is not likely to be a bottleneck, it will not be replicated.

Name Lookup

We must revisit the issue of object naming with the advent of object components. Recall that we have hidden a location-based naming scheme within an RcapGlobId by concatenating the IP address of the object's location with the object type and number. It is clear, however, that as we move to a distributed implementation of objects, we must change our technique of object naming. While some sort of global name must still be assigned to the logical object, we must resolve this name into one of the names of the different physical components depending on the location of the access. This level of indirection can be hidden within the location and naming service.

In this new framework, we must institute Regional Location and Naming Servers, RLNS. A RLNS is placed in each region and it holds mappings of global object names to the names of local object components placed in its region. The local LNS residing in the daemon now becomes a cache of lookups. An object's global name is simply the RcapGlobId of the initial component of the object. If a new component is made, the new component's RLNS is given the mapping of the global name to the regional object component's local ID. When a client makes a request, the object is looked up at the local daemon's LNS; if an entry is not found there, then the RLNS is queried. If the RLNS returns with a mapping, then that information is cached at the LNS, using an LRU replacement policy. If no mapping is returned, then the global name is used in a similar fashion to how it is used our current implementation

.

Global Name	Object Lookup
homeIP/object/instance	localIP/object/instance
homeIP/object/instance	localIP/object/instance
homeIP/object/instance	localIP/object/instance
homeIP/object/instance	localIP/object/instance

FIGURE 10: Lookup table in the RLNS and LNS

When a component of an object is destroyed, then its entry must be purged from the RLNS. As long as there is robust error handling to service the case when a component no longer exists, the cached entries in the LNS lookup table need not be removed. In other words, a daemon may send a request to the cached address, get back an error message saying the object does not exist, and then purge the entry and use the global address.

Connection Setup

In our proposed implementation, all components of a target set object are aware of all channels sending to that target set. During channel creation, a single component of the channel object is instantiated. The channel object is then replicated during connection setup, with a component placed in every region where components of the target set object exist. Connection setup is modified as follows, with the initial component of the channel object coordinating each step.

1. The channel object contacts the local target set object component to get a list of the current members and a list of any other components of the target set object.
2. For each component of the target set object, a component of the channel object is placed in that region.
    a. The new local component of the channel object queries that region's target set object to get its members and its list of other target set components. At this point, the local target set object knows that a new channel is attempting establishment, and any subsequent joins to that component of the target set object will result in an attach request being sent directly to this new local component of the channel object.
    b. The local target set object's list of components is passed back to the initial component of the channel object. Then at the primary site, this list is merged with all other components previously received.
3. When all components of the channel object have been created, the initial component begins the preparation phase and signals each other component to begin the process as well. In this system, the preparation phase can be implemented in parallel with connections routed regionally, and cross-region attachment points mutually agreed upon by the routing servers.
4. Once this phase completes, the results are sent to the source node to begin establishment. The distributed establishment phase is implemented as before.

Dynamic Join

In a move to a more scalable solution, we must implement receiver-initiated attachment as outlined in Section 3.1.2. As subsequent receivers join a local component of a target set object, the local component of the channel object is informed. Only that component of the channel object coordinates attachment of the new receiver, and no other components of the channel object must be contacted[6]. In this new framework, establishment events can be serialized regionally; when an event that spans the entire multicast tree such as DTM crosses regional borders, it is serialized by the local component of the channel object. For example, if a receiver attachment is currently being processed, and a DTM event arrives from another region, the DTM event will be enqueued until the attachment completes. Serialization of these events on a regional basis results in an increased concurrence of establishment events.

Dynamic Traffic Management

A component of a channel object must be able to serialize establishment events in its region. The procedures for dynamic traffic management are relatively unaffected by the distribution of objects, with one exception; the forward pass of the prepare phase must halt prior to entering a new region. The new local component of the channel object must be informed of the DTM event arriving in its region so that it may serialize and coordinate events in its region. Similarly, when the final phase of DTM completes in a region, the channel object component is informed that the event has finished.

Dynamic Leave and Final Teardown

If a receiver leaving a target set is the last member of that component of the corresponding target set object, then the component is removed. All other components of the target set object are informed so that they may flush that component's name and address from their list. In addition, the RLNS is informed, so that it may flush that entry for the target set object. This process, however, does not occur if the receiver leaves the initial component of the target set object. In this case, the component remains until the target set is deleted, so that we always have at least one instantiation of the object that is referenced by the global ID.

Dynamic detachment of a receiver from a channel only involves the local component of that channel's object. Again, if this is the last receiver in this region, and this is not the initial region, then the component of the channel object is removed.

Teardown results in the destruction of all components of the channel object other than the initial component. Therefore, if a channel is in a passive state, there exists only one component of the object.

5.3.4 Analysis and alternate proposal

Although the outlined changes solve several scaling problems, one still remains: all components of the target set object must know about all channels sending to the corresponding target set.

An alternative design would have each component of a target set object know only about channel objects that had initial components in their region. In this scenario, all establishment events would have to be funneled through this initial (primary) component of the channel object, and be dispatched to the affected component of the channel for servicing. Joins would be broadcast to all components of the target set object, so that each component could contact the subset of recognized channels. This second design has the drawback of increased communication. It also is unclear if the amount of additional data required in the first approach would ever scale to such a point as to require this optimization. We anticipate that the issue of hot spots will have the largest impact on

6. We may wish to mandate that attachment points must be in the same region as the joining receiver.

the scaling of the system. This latter design idea reintroduces the hot spot at the channel object by forcing all establishment events through its initial component.

6.0 Comparison with other approaches

It is a valuable exercise to compare the scalability of this design with those of other approaches proposed within the network research community. In this section we will examine two protocols designed to support multi-party applications, ST-II and RSVP. ST-II is a network layer protocol that provides not only data delivery and routing but also resource reservation, whereas RSVP is designed simply as a resource reservation protocol. It is expected that RSVP will be deployed in conjunction with the new version of IP Multicast, which will supply the data transport and routing functions.

6.1 ST-II Signaling (SCMP)

ST-II is a connection oriented network protocol that includes mechanisms for connection setup, teardown, connection modification, data delivery, and routing. ST-II is a general specification that has generated several implementations. This section will begin with an examination of ST-II's signalling protocol (SCMP) as specified, and then move on to a discussion of the scalable extensions added to the Heidelberg Transport System's (HeiTS) implementation of SCMP.

In ST-II, resource reservation is based upon a *flowspec* that contains a rich set of quality of service parameters, which not only characterize the service requested by the receiver but also model the source's traffic. This is in contrast with RCAP2, which makes a distinction between a source's traffic specification and a receiver's performance requirements by having them specified separately. We make a distinction between those parameters that are specifically under the control of the sender and those that simply reflect how the data are to be received. The flowspec as listed in the ST-II specification consists of a large set of parameters. This set includes such common indices as throughput and delay, as well as several less clearly defined parameters such as *DutyFactor* and *Reliability* [DelHerHofSch94]. Several implementors of ST-II have chosen to allow the use of only some subset of these parameters.

All establishment events are sender-oriented. During establishment, a route is dynamically generated from the source to the set of receivers, and resources are allocated[7] on the way. If it is discovered during the resource reservation process that there are not enough resources to support the flowspec, then the flowspec is modified to reflect the amount of resources that can be supplied on that particular path. In addition, the flowspec may also be modified to reflect the delay bound seen thus far in that path from the source. When a message containing a flowspec reaches a *target* (i.e., a receiver), this can either accept or reject the connection. If the connection is accepted, then the flowspec is sent back to the source. All targets must adhere to the same flowspec; in other words, there must be a homogeneous data rate along the multicast connection. After the flowspec replies have been received, the source must decide the final flowspec. It has the choice of either picking the "lowest common denominator" of the flowspecs and issuing a CHANGE request, or of disconnecting those targets from the connection with reduced flow specifications. Suite 2, on the other hand fixes the traffic specification prior to establishment, and does not require this further negotiation phase. As we stated earlier in this document, we suggest the use of hierarchical encoding and

7. It is important to note that the ST-II protocol specification does not describe how resources are to be reserved or scheduled.

multiple target sets to handle the issue of heterogeneous receivers. As illustrated in Section 5.1.3, we can handle the problem of bottleneck links without the need for this additional negotiation phase.

It is implicit within the description of the ST-II establishment process that partial establishment semantics are supported. ST-II also supports dynamic attachment and detachment of receivers. All establishment is sender-oriented. A *target* must request attachment through an out-of-band message. The source then generates a CONNECT message to add that target to the connection. This attachment process follows the description above, with the source specifying a flowspec that may be modified as it progresses towards the target. The target may then accept or reject the connection.

ST-II's sender-oriented approach has been the center of much criticism [MiEsShZh94], [DelHerHofSch93]. By having all establishment attempts funneled through one point, the source becomes a bottleneck and a scaling limitation. The Heidelberg Transport System's (HeiTS) implementation has addressed this problem by adding support for receiver-initiated attachment and providing it in two extensions.

The first extension, "join stream at router", begins with a target sending a request message to the source. If that message intersects with the existing multicast tree, the request is serviced by that node of the tree. Connection establishment then progresses from that point in the tree back to the receiver. If the target accepts the flowspec, then the source can be optionally informed.

The second extension, "create path backwards", progresses from the target toward the source. The initial flowspec must be provided out-of-band or chosen by the target. This latter approach can lead to a mismatch between resource allocations. The authors of [DelHerHofSch93] suggest that "a small set of encoding and data format helps to solve the problem". However, it is difficult to restrict the number of encoding and data formats if we wish to support a wide variety of applications, which again touches on the problem of scalability. Suite 2 handles this problem by having the channel object involved in the attachment process. All components of this object contain the traffic specification of the source so that this information is available during attachment. It is important to note that, if we do not employ the approach outlined in Section 5.3, our design has simply replaced the existing bottleneck with one at the channel object.

Resource sharing can be supported in ST-II through the *stream groups* mechanism; however, algorithms must still be developed to employ this feature. ST-II is obviously less scalable without this feature for the reasons discussed in Section 5.1.2.

On a final note, we must address the issue of fault management. Both ST-II and Suite 2 are connection-oriented. As such, they are more vulnerable to faults along the path of a connection. After a fault occurs, there will be some disruption in the service while the route is rebuilt. Connectionless protocols must also re-route around failures, but sources may continue sending during this process.

6.2 RSVP

Unlike ST-II, RSVP is simply a reservation protocol. As such, it distributes resource requirements and negotiates quality of service values. It does not perform any resource reservation or admission control. In addition, it is assumed that both data delivery and routing will be accomplished by companion protocols such as IP Multicast. Unlike both ST-II and the Tenet Suite 2 Protocols, RSVP uses a MxN connection as its basic paradigm. This approach is supported by IP Multicast, which allows for several sources to send to a single multicast address.

RSVP uses a receiver-oriented reservation process [ZhaDeer93]. Receivers supply a *flowspec* that contains both the requested data rate and the performance requirements for that receiver. Resources are negotiated for the receiver on the basis of this flowspec. In addition, RSVP supports a *packet filter* which indicates the way in which a receiver wishes to use the reserved resources. In the initial design, three filter types were identified: the null filter, a fixed filter, and a dynamic filter. If the receiver does not use a filter, then it will receive all data sent from any source to the multicast address. Using a fixed filter, a receiver may specify a set of sources from which it wishes to receive data. The packets from these sources may use the resources that have been allocated on behalf of the receiver. A dynamic filter is used if the receiver wishes to "switch channels", thereby dynamically choosing which source can use the resources. The dynamic filter option is currently labeled "experimental" in the RSVP specification [BrZhEsHeJa95]. Given the basic MxN paradigm, there is some complexity involved in negotiating resources on common sub-paths for receivers using different packet filters.

We believe that RSVP's receiver-oriented approach is more scalable than a strict sender-oriented approach, but we also feel that a hybrid of the two more naturally supports multi-party applications. Senders can best characterize their own traffic, so we feel that in a number of applications it is reasonable that the initial establishment of the data stream be initiated at the sender. In addition, receivers should have the ability to dynamically join an ongoing connection. This process should be receiver-initiated, and should not bottleneck at the sender. RSVP's proposed receiver-oriented design leaves some open questions. For example, it is unclear how the data are to be formatted at the source in order to supply the varying rates specified by the collection of heterogeneous receivers. The use of hierarchical encoding has been suggested in [ZhaDeer93], but this approach implies that the data encoding format is visible to the filtering at the network layer. In addition, does the filtering code in the network recognize all encoding formats, or is this information somehow user-supplied [DelHerVogWol93]? In Suite 2, we avoid these issues by suggesting that separate channels be set up for each encoded layer. This technique makes the support for multiple data rates invisible to the network. Our approach does require the receiver to reassemble the data arriving on different channels. In RSVP, this layered data would be delivered with the same stream, and thus may be somewhat easier to reassemble and synchronize.

RSVP supports a connectionless service supplied by IP Multicast, and uses *soft state* to describe the reserved resources. This soft state must be periodically refreshed or it will time out and be discarded. Receivers must send out *reservation* messages containing their requested flow specs and packet filters. The RSVP designers have tried to reduce the impact of these periodic multicasts by coalescing messages along the multicast route. Before the reservation messages from multiple receivers are forwarded toward each source, they are combined at branch points into a single bounding flowspec.

In addition to requiring that receivers generate reservation messages, sources must send out *path* messages. A path message contains the source's flowspec and an F-flag indicating if it will allow filtered reservations on its data. These periodic path and reservation messages provide a technique for rendezvous in this protocol which allows RSVP to more easily adapt to faults in the network. During the period of recovery, prior to path and reservation message rendezvous, data will be delivered unfiltered in a best-effort fashion. If receivers have specified filters, they may receive a barrage of unwanted data during this process.

In summary, by taking a connectionless approach using soft state, the protocol is more tolerant to faults in the network. In addition, RSVP's receiver-oriented approach is more scalable than ST-II's original sender-oriented service. It is not clear, however, if this is more scalable than Suite 2's hybrid approach, which allows both sender- and receiver-initiated actions. Finally, although RSVP is a scalable reservation protocol, it does not attempt to support the type of guaranteed-performance service that is the target of the Tenet Protocols.

7.0 Summary

We have provided a detailed examination of the Tenet Suite 2 Protocols, looking in particular at the issue of scalability in the RCAP2 signaling protocol. Suite 2 addresses the problem of supplying guaranteed service over a packet-switched network for multi-party applications. Within this framework, we have supplied a scalable design that supports large heterogeneous internetworks, many connections, and many receivers on a single connection. We support heterogeneous networks by supporting multiple scheduling algorithms and by fully distributing the admission control process. By adding such mechanisms as resource sharing and dynamic traffic management, we can support more efficient use of network resources, allowing for more real-time connections to be created. Finally, the design can support heterogeneous receivers through the use of hierarchical encoding and multiple concentric target sets.

Several implementation decisions we made in building the prototype of Suite 2 did not scale. The most notable of these was the decision to have only a single representation of each object instance. We have sketched an alternative implementation that both scales well and supports the original RCAP2 design. Through this exercise we have shown that, while there may be scaling flaws in our initial prototype implementation of RCAP2, the design is much more scalable.

Acknowledgments

I would like to thank my fellow members of the Tenet Suite 2 Group for their support through the years during the design and implementation of these protocols. I have learned a great deal each of them. In particular, I would like to thank Mark Moran for directing me toward the topic of scalabily and for discussing initial ideas with me to solve some of the scaling problems in our prototype.

Finally, I would like to thank my advisor, Professor Domenico Ferrari, for his guidance, insight and kindness.

References

[BetFerGupHe95] Ricardo Bettati, Domenico Ferrari, Amit Gupta, Wendy Heffner, Wingwai Howe, Mark Moran, Quyen Nguyen, Raj Yavatkar. Connection Establishment for Multi-Party Real-Time Commuication. *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 255-270, Durham, New Hampshire, April 1995

[BierNonn95] Ernst Biersack, Jorg Nonnenmacher. WAVE: A New Multicast Routing Algorithm for Static and Dynamic Multicast Groups. *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 243-254, Durham, New Hampshire, April 1995

[BolCreGar95] Jean-Chrysostome Bolot, Hugues Crepin, Andres Vega Garcia. Analysis of Audio Packet Loss in the Internet. *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 163-174, Durham, New Hampshire, April 1995

[BrZhEsHeJa95] R. Braden, L.Zhang, D. Estrin, S. Herzog, and S.Jamin. Resource ReSer-Vation Protocol (RSVP) - Version 1 functional specification, Internet Draft, April 12, 1995

[DelHerHofSch94] Luca Delgrossi, Ralf Guido Herrtwich, and Frank Oliver Hoffmann. An Implementation of ST-II for the Heidelberg Transport System, *Internetworking - Research and Experience*, Vol. 5, 1994

[DelHerVogWol93] Luca Delgrossi, Ralf Guido Herrtwich, Carsten Vogt and Lars C. Wolf. Reservation protocols for internetworks: A Comparison of ST-II and RSVP, *Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 199-207, Lancaster, U.K., 1993

[DelHerHofSch93] Luca Delgrossi, Ralf Guido Herrtwich, Frank Oliver Hoffmann and Sibylle Schaller. Receiver-Initiated Communication with ST-II, preliminary version, Sept. 1993,

[EffMul93] Wolfgang Effelsberg and Eberhard Muller-Menrad. Dynamic Join and Leave for Real-Time Multicast. International Computer Science Institute, TR-93-056, September 1993

[FerGup93] Domenico Ferrari and Amit Gupta. Resource partitioning in Real-Time Communication. *Proceedings of the IEEE Symposium on Global Data Networking*, pages 128-135, Cairo, Egypt, December 1993

[FerGupVen95] Domenico Ferrari, Amit Gupta, and Giorgio Ventre. Distributed advance reservations of real-time connections. *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 15-26, Durham, New Hampshire, April 1995

[Ferrari92] Domenico Ferrari. Real-Time Communication in an Internetwork, *Journal of High Speed Networks*, pages 79-103 vol. 1 no. 1, 1992

[FerVer90] D. Ferrari and D. C. Verma, A Scheme for Real-Time Channel Establishment in Wide-Area Netwroks, *IEEE Journal: Selected Areas in Communications, SAC-8*, pages 368-379, April 1990

[GroKesTse95] M. Grossglauser, S. Keshav and D. Tse. The Case Against Varriable Bit Rate Service. *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video*, Pages 307-310,Durham, New Hampshire, April 1995

[GuHoMoNg95] Amit Gupta, Wingwai Howe, Mark Moran, and Quyen Nguyen. Resource sharing in multi-party realtime communication. *Proceedings of INFOCOM 95*, Boston MA, April 1995

[MiEsShZh94] Danny J. Mitzel, Deborah Estrin, Scott Shenker and Lixia Zhang, An Architectural Comparison of ST-II and RSVP, *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, Toronto, Canada, June 1994

[Moran95] Private discussions with Mark Moran on his thesis topic, U.C. Berkeley, CA and *Industrial Liaison Program* talk, Berkeley, CA, March 1995

[OppDal83] D.C. Oppen and Y.K. Dalal. The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment., *ACM Transactions on Office Information Systems* 1(3):230-253, July 1983

[ParBan94] C. Parris and A. Banerjea. An Iinvestigation into Fault Recovery in Guaranteed Performance Service Connections. *Proceedings of SUPERCOMM/ICC'94*, New Orleans, LA, pages 175-181, March 1994

[ParGal93] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flowcontrol in integrated services networks: The single node case," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 344--357, June 1993.

[SchBirNeed84] M.C. Schroeder, A. D. Birrell, and R. M. Needham. Experience with Grapevine: The Growth of a Distributed System. *ACM Transactions on Computer Systems* 2(10):3-23, February 1984

[Widyono94] R. Widyono. The Design and Evaluation of Routing Algorithms for Real-time Channels. International Computer Science Institute, TR-94-024, June 1994

[YavPaiFin94] R. Yavatkar, P. Pai and R. Finkel. A Reservation-based CSMA Protocol for Integrated Manufacturing Networks. *IEEE Transactions on Systems, Man and Cybernetics* 1994

[ZhaDeer93] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Networks Magazine*, 30(9):8-18, September 1993

[ZhaKni95a] Hui Zhang, Edward W. Knightly. Traffic Characterization and Switch Utilization Using a Determenistic Bounding Interval Dependent Traffic Model. *Proceedings of IEEE INFOCOM'95*, Boston, MA, April 1995

[ZhaKni95b] Hui Zhang, Edward W. Knightly. A New Approach to Support Delay-Sensitive VBR Video in Packet-Switched Networks. *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video*, Pages 275-286, Durham, New Hampshire, April 1995

[Zhang93] Hui Zhang. Service Disciplines for Packet-Switching Integrated Service Networks. University of California at Berkeley. PhD Dissertation 1993