RSVP: A New Resource ReSerVation Protocol

Lixia Zhang[1], Steve Deering[1], Deborah Estrin[2], Scott Shenker[1], Daniel Zappala[3]

{lixia, deering, shenker}@parc.xerox.com, {estrin, zappala}@usc.edu

ACCEPTED BY IEEE NETWORK MAGAZINE

[1]Xerox PARC, 3333 Coyote Hill Road, Palo Alto, CA 94304.

[2]Information Sciences Institute and Computer Science Department, University of Southern California.

[3]Computer Science Department, University of Southern California, Los Angeles, CA 90089.

1 Introduction

The current Internet architecture, as embodied in the IP network protocol, offers a very simple service model: point-to-point best-effort service. In recent years, several new classes of distributed applications have been developed, such as remote video, multimedia conferencing, data fusion, visualization, and virtual reality. It is becoming increasingly clear that the Internet's primitive service model is inadequate for these new applications; this inadequacy stems from the failure of the point-to-point best-effort service model to address two application requirements. First, many of these applications are very sensitive to the quality of service their packets receive. For a network to deliver the appropriate quality of service, it must go beyond the best-effort service model and allow flows (which is the generic term we will use to identify data traffic streams in the network) to reserve network resources. Second, these new applications are not solely point-to-point, with a single sender and a single receiver of data; instead, these applications can often be multipoint-to-multipoint with several senders, and several receivers, of data. Multipoint-to-multipoint communication occurs, for example, in multiparty conferencing where each participant is both a sender and a receiver of data, and also in remote learning applications, although in this case there are typically many more receivers than senders.

In recent years there has been a flurry of research activity devoted to the development of new network architectures and service models to accommodate these new application requirements. Even though there are rather fundamental differences between the various proposed architectures, there is widespread agreement that any new architecture capable of accommodating multicast and a variety of qualities of service can be divided into five distinct components, which we identify and describe below.

Flow Specification The network and the various data flows need a common language so that a source can tell the network about the traffic characteristics of its flow, and the network can in turn specify the quality of service to be delivered to that flow. Thus, the first component of this new architecture is a flow specification, or *flowspec*, which describes the characteristics of both the traffic stream sent by the source, and also the service requirements of the application. In some sense the flowspec is the central component of the architecture, since it embodies the service interface that applications will interact with; the details of all of the other components of the architecture are hidden from applications. Two proposals for a flowspec are described in References [14, 17].

Routing The network must decide how to transport packets from the source to the receiver or, in the case of multicast, receivers of the flow. Thus, the second component of the architecture is a routing protocol that can provide quality unicast and multicast paths. There are many approaches to unicast routing; References [1, 5, 17] describe different approaches to multicast routing. None of the current proposals have yet dealt sufficiently with the interaction between routing and quality of service constraints; that is the subject of future research.

Resource Reservation In order for the network to deliver to a particular flow a quantitatively specified quality of service, like a bound on delay, it is usually necessary for the network to set aside certain resources, such as a share of bandwidth or a number of buffers, for that flow. This ability to create and maintain resource reservations on each link along the transport path is the third component of the architecture. References [7, 17] describe two approaches to resource reservation; in this article we describe another.

Admission Control Because a network's resources are finite, it cannot grant all resource reservation requests. In order to maintain the network load at a level where all quality of service commitments can be met, the network architecture must contain an admission control algorithm; this algorithm determines which reservation requests to grant and which to deny, and thereby maintains the network load at an appropriate level. References [10] and [13] describe two admission control algorithms.

Packet Scheduling After every packet transmission, a network switch must decide whether and which packet to transmit next. It is the packet scheduling algorithm which controls this decision. The packet scheduling algorithm lies at the heart of any network architecture because it determines which qualities of service the network to can provide. There are many proposed packet scheduling algorithms; see References [4, 6, 8, 11, 12] for a few examples.

In this article we present our proposal for the third component of the architecture, a new resource ReSerVation Protocol, RSVP. Similar to previous work on resource reservation protocols (e.g. ST-II [17]), RSVP is a simplex protocol, i.e. it reserves resources in one direction. However, several novel features in the RSVP design lead to the unique flexibility and scalability of the protocol. RSVP is receiver-oriented, in that the receiver of the data flow is responsible for the initiation of the resource reservation. This design decision enables RSVP to accommodate heterogeneous receivers in a multicast group; specifically, each receiver may reserve a different amount of resources, may receive different data streams sent to the same multicast group, and may "switch channels" from time to time (that is, change which data streams it wishes to receive) without changing its reservation. RSVP also provides several reservation styles that allow applications to specify how reservations for the same multicast group should be aggregated at the intermediate switches; this feature results in more efficient utilization of network resources. Finally, by using "soft-state" in the switches, RSVP supports dynamic membership changes and automatically adapts to routing changes. These features enable RSVP to deal gracefully and efficiently with large multicast groups. While the motivation for RSVP arose within the Internet context, our design is intended to be fully general.

This article has 9 sections. We first list our design goals (Section 2), and then discuss the basic design principles used to meet these goals (Section 3). Section 4 contains a more detailed description of the protocol operation, and Section 5 describes how the protocol would work in a simple example. Section 6 describes the current state of our RSVP implementation. We delay consideration of related work until Section 7, and follow that with a discussion of unresolved issues in Section 8. Finally, in Section 9, we conclude with a brief summary.

2 RSVP Design Goals

In the traditional point-to-point case, one obvious reservation paradigm would have the sender transmit a reservation request towards the receiver, with the switches along the path either admitting or rejecting the flow. For the point-to-multipoint case, one may trivially extend this paradigm to have the sender transmit the reservation request along a multicast routing tree to each of the receivers. When we have multipoint-to-multipoint data transmissions, the straightforward extension of this paradigm would be to have each sender transmit a reservation request along its own multicast tree to each receiver. However, the special properties of having multiple, heterogeneous, receivers and/or multiple senders pose serious challenges that are not addressed by this simple extension of the basic reservation paradigm. We outline these various challenges below and detail how these challenges are not met by the strawman proposal of straightforwardly extending the basic paradigm. In the process, we identify the seven goals which guided the design of RSVP.

In a wide area internetwork such as the Internet, receivers, as well as the paths used to reach the receivers, can have very different properties from one another. In particular, one must not assume that all the receivers of a multicast group possess the same capacity for processing incoming data, nor even necessarily desire or require the same quality of service from the network. For instance, a source may be sending a layered encoding of a video signal; it is possible that certain receivers, which are doing the decoding in software, would only have sufficient processing power to decode the low-resolution signal while those receivers with hardware decoding, or more processing power, could decode the entire signal. Furthermore, the paths used to reach all the receivers may not have the same capacity; in the layered encoding example above, certain receivers might only have low-bandwidth paths between them and the source and so could only receive the low-resolution signal. The strawman proposal above is incapable of dealing with the receivers individually, and so cannot address these heterogeneous needs. Therefore, our first design goal for RSVP is to provide the ability for heterogeneous receivers to make reservations specifically tailored to their own needs.

The presence of multiple receivers raises another issue; the membership in a multicast group can be dynamic. The strawman proposal would have to reinitiate the reservation protocol every time a new member joined or an existing member left the multicast group. Reinitiation of the reservation protocol is particularly burdensome for large groups because the larger the group size, the more frequent are changes in group membership. So our second design goal for RSVP is to deal gracefully with changes in the multicast group membership.

The strawman proposal deals with multiple senders by having each sender make an independent resource reservation along its own multicast routing tree. This approach results in resources being reserved along multiple, independent trees, even though the branches of different trees often share common links. This may be appropriate for some applications, but in other cases this duplication can lead to a significant wasting of resources. For example, in an audio conference with several people, there is usually only one person, or at most a few people, talking at any one time because of the normal dynamics of human conversation. Thus, instead of reserving enough bandwidth for every potential speaker to speak simultaneously, in many circumstances it would be adequate to reserve only enough network resources to handle a few simultaneous audio channels. Our third design goal is for RSVP to allow end users to specify their application needs so that the aggregate resources reserved for a multicast group can more accurately reflect the resources actually needed by that group.

Furthermore, in a multiparty conference, a receiver may only wish to, or may only be able to, watch one or a few other participants at a time, but would like the possibility of switching among various participants. The simple approach of delivering the data streams from all the sources and then dropping the undesired ones at the receiver does not address network resource usage considerations (e.g., efficient use of limited bandwidth, or reducing the charges incurred for bandwidth usage). A receiver should be able to control which packets are carried on its reserved resources, not only what gets displayed on its local screen. Moreover, a receiver should be able to switch among sources without the risk of having the change request denied, as could occur if a new reservation request had to be submitted in order to "switch channels". Our fourth design goal for RSVP is to enable this channel changing feature.

RSVP is not a routing protocol, and should avoid replicating any routing functions. RSVP's task is to establish and maintain resource reservations over a path or a distribution tree, independent of how the path or tree was created. In a large internetwork with a volatile topology and load, these routes may change from time to time. Adapting to such changes in topology and load is the explicit job of the routing protocol and it would be expensive and complicating to replicate the function in RSVP. At the same time, however, RSVP should be able to cope with the resulting routing changes. Our fifth design goal is that RSVP should deal gracefully with such changes in routes, automatically reestablishing the resource reservations along the new paths as long as adequate resources are available.

The strawman proposal does not deal gracefully with changes in routes, because there is no mechanism to discover the change and trigger a new resource reservation request. One could introduce such a mechanism by having each source periodically *refresh* its reservation over the multicast routing tree. However, in large multicast groups such refreshing would lead to S messages arriving at every receiver during every refresh period, where S is the number of sources. Our sixth design goal is to control protocol overhead; by this we mean both avoiding the explosion in protocol overhead when group size gets large, and also incorporating tunable parameters so that that the amount of protocol overhead can be adjusted.

Our last design goal is not specific to the problem at hand but rather is a general matter of modular design; we hope to make the general design of RSVP relatively independent of the other architectural components listed in Section 1. Clearly a particular implementation of RSVP will be tied quite closely to the flowspec and interfaces used by the routing and admission control algorithms. However, the general protocol design should be independent of these. In particular, our protocol should be capable of establishing reservations across networks that implement different routing algorithms, such as IP unicast routing, IP multicast routing [5], the recently proposed CBT (Core-Based Tree) multicast routing [1], or some future routing protocols. This design goal makes RSVP deployable in many contexts. However, for optimally efficient routing decisions, routing selection and resource reservation should be integrated so that the choice of route can depend on the quality of service requested, and so that the stability of the route can be maintained over the duration of the reservation. Such an integration would lead to more coordination between the choice of which resources to reserve and the mechanics of establishing the reservation (which is RSVP's main focus). This integration is something that requires further research.

Thus, in summary, we have identified seven important design goals, which we list in Figure 1. RSVP is primarily a vehicle used by applications to communicate their requirements to the network in a robust and efficient way, independent of what the specific requirements are. RSVP delivers resource reservation requests to the relevant switches, but plays no other role in providing network services. Thus, RSVP can communicate the requirements for, but does not directly provide for, a wide range of network services. For instance, the synchronization requirements of flows or the need for reliable multicast delivery could be expressed in the flowspec that is distributed by RSVP and then realized by the switches. Similarly, the flowspec could also carry around information about advance reservations (reservations made for a future time) and preemptable reservations (reservations that a receiver is willing to have preempted). RSVP is capable for supporting the delivery of these and other services whenever these network services rely only on state being established at the individual switches along the paths determined by the routing algorithm. Thus, while we have described RSVP as a resource reservation protocol, it can be seen more generally as a *switch state establishment* protocol.

3 Basic Design Principles

To achieve the seven design goals listed in Figure 1, we used six basic design principles which we now describe. These design principles are listed in Figure 2.

3.1 Receiver-Initiated Reservation

The strawman proposal discussed in the previous section and all existing resource reservation protocols are designed around the principle that the data source initiates the reservation request. In contrast, RSVP adopts a novel *receiver-initiated* design principle; receivers choose the level of resources reserved and are responsible for initiating and keeping the reservation active as long as they want to receive the data. We describe the motivation for this receiver-initiated approach below.

A source can always transmit data, whether or not adequate resources exist in the network to deliver the data. It is the receiver who knows its own capacity limitations; furthermore, the receiver is the only one who experiences, and thus who is directly concerned with, the quality of service experienced by the incoming packets. Additionally, if network charging is deployed in the future, the receiver would likely be the party paying for the requested quality of service. Thus, it should be the receiver who decides what resources should be reserved.

One could imagine having the receivers send this information to the source and then having the source use this information in sending out the reservation request. To handle heterogeneous requests, however, this approach would require that the sender bundle all requests together to pass to the network, so that the network can figure out, according to the location of individual receivers, how much resource needs to be reserved on which links. For large multicast groups, this will likely cause a multicast implosion at the sender. This implosion problem becomes more serious when the multicast group membership changes dynamically and the reservation has to be periodically renewed. Consider, as an extreme example, a cable TV firm with broadcasting several channels of programs; while there are relatively few sources, there are perhaps hundreds of thousands of receivers, each of which can watch only one or a few channels at a time. In the strawman proposal, whenever any individual receiver wants to switch between channels, it sends a message to the source. In this case, where there are many receivers and frequent switching between channels, each source has to accommodate an deluge of change requests. However, this overhead is clearly superfluous, since we expect the resulting broadcast pattern to change relatively slowly because the resulting multicast trees will likely be relatively stable except near the leaf nodes. In Section 4 we show how our receiver-initiated design accommodates heterogeneity among group members yet avoids such multicast implosion.

The idea of the receiver-initiated approach was first inspired by Deering's work on IP multicast routing [5]. The IP multicast routing protocol treats senders and receivers separately. A sender sends to a multicast group in exactly the same way as it sends to a single receiver; it merely puts in each packet a multicast group address in place of a host address. The multicast group membership is defined as the group of receivers. Deering's multicast routing design can be considered a receiver-initiated approach, in the sense that each receiver individually decides to join or leave the group without affecting other receivers in the group, or affecting sources that send to the group. It is the routing protocol that takes the responsibility of forwarding all multicast data packets to all the current members in the group. Analogous to our argument that a sender does not care whether adequate resources are available, a sender to a multicast group does not necessarily know who is currently a member of the multicast group (i.e., receiving the data); in particular, it may not be a member of the multicast group itself.

3.2 Separating Reservation from Packet Filtering

A resource reservation at a switch assigns certain resources (buffers, bandwidth, etc.) to the entity making the reservation. A distinction that is rarely made, which will be crucial to our ability to meet our design goals, is that the resource reservation does not determine which packets can use the resources, but merely specifies *what amount* of resources is reserved for *whom*. Notice that the *whom* refers not to which packets can use the reserved resources, but rather specifies which entity *controls* the resources. There is a separate function, called a packet *filter*, which selects those packets that can use the resources; this filter is set by the reserving entity. Moreover, the filter can be changed *without* changing the amount of reserved resources. Thus, one of the important design principles in RSVP is that we allow this filter to be *dynamic*; that is, the receiver can change it during the course of the reservation. This distinction between the reservation and the filter will enable us to offer several different reservation styles, which we now describe.

3.3 Providing Different Reservation Styles

As we discussed briefly in Section 2 above, the service requirements of multicast applications dictate how the reservation requests from individual receivers should be aggregated inside the network. For example, as we discussed in Section 2, the typical dynamics of human verbal interaction results in only one or a few people talking at any one time; thus, in many conferencing situations it would be feasible to have all senders of audio signals to a conference share the same set of reserved resources, where these resources were sufficient for a small number of simultaneous audio streams. In contrast, there are no analogous limitations on video signals. Therefore, if the conferencing application also includes video then enough resources must be reserved for the number of video streams one desires to watch simultaneously. As in the usual multicast paradigm, if two receivers downstream of a particular link are watching the same video stream for the lifetime of the application (e.g. when attending a remote lecture), only a single reservation need be made on this link to accommodate their needs. However, if these two receivers wish to occasionally switch among the senders during the application lifetime (e.g. when participating in a distributed group meeting), then separate reservations must be maintained. To support different needs of various applications, while making the most efficient use of network resources, RSVP defines different *reservation styles* which indicate how intermediate switches should aggregate reservation requests from receivers in the same multicast group. Currently there are three reservation styles: *no-filter*, *fixed-filter*, and *dynamic-filter*. We now describe these filter styles; for the sake of brevity we will identify applications only by their multicast address, although in the current Internet context a multicast application may be identified by the IP multicast address plus destination port number.

When a receiver makes a resource reservation for a multicast application, it can specify whether or not a data source filter is to be used. If there is no filter, then any packets destined for that multicast group may use the reserved resources (although some enforcement mechanism is needed to make sure that the aggregate stream does not use more than the reserved amount; we will not discuss enforcement mechanisms here). For example, the audio conference described above would use a *no-filter* reservation, so that a single reserved pipe can be used by whomever is speaking at the moment. If source filtering is needed, the filter is specified by a list of sources (again, in the Internet context a data source can be specified by the source host address plus source port number, but we will only refer to the source host address in this exposition). Only the packets from the specified sources can use the reserved resources. Filtered reservations will be used to forward individual images in video conferencing, enabling participants to reserve resources for particular video streams.

A filtered reservation can be either fixed or dynamic. A *fixed-filter* reservation means that for the duration of the reservation, the receiver will receive data only from the sources listed in the original reservation request. A *dynamic-filter* reservation allows a receiver to change its filter to different sources over time.

In order to illustrate how intermediate nodes use these reservation styles to aggregate reservation requests, consider the case where several receivers in the same multicast group make *fixed-filter* reservations over a common link. These reservations may be shared if the source lists overlap, because the reservation will never be changed. Thus, only a single pipe (with the largest amount of resources from all the requests) will be reserved for each source even when there are multiple requests. Such aggregation can occur when members of a multicast application all listen or watch the same audio or video signals, as in the case of a multicast lecture. Reservations using the *no-filter* style can also be aggregated in this manner, because if a receiver does not discriminate between individual sources, it cannot switch among the sources either.

If a receiver expects to switch among different sources from time to time, it must make a *dynamic-filter* reservation to avoid affecting the reception of other receivers in the same multicast application. Because the receiver can change the list of sources in the filter at any time during the course of the reservation, the intermediate nodes cannot aggregate reservations of this style. In fact, this separation between the resource reservation and the filter is one of the key facets of RSVP; the resource reservation controls how much bandwidth is reserved, while the filter controls which packets can utilize that bandwidth. In the *dynamic-filter* reservation case, each receiver requests enough bandwidth for the maximum number of incoming streams it can handle at once, and the network must reserve enough resources to handle the worst case when all the receivers that requested dynamic filter reservations take input from different sources, even though several receivers may actually tune to the same source(s) from time to time. However, note that the total amount of dynamic filter reservations made over any link should be limited to the amount of bandwidth needed to forward data from all the upstream sources.

In summary, having several different reservation styles allows intermediate switches to decide how individual reservation requests for the same multicast group can be efficiently merged. The dynamic filter reservation style allows receivers to change channels. Thus, we have met design goals 3 and 4. So far RSVP has defined three reservation styles; other styles may be identified as new multicast applications, with different needs, are developed.

3.4 Maintaining "Soft-State" in the Network

The typical multipoint-to-multipoint applications we have considered are rather long-lived. Over the lifetime of such an application, it is quite possible that new members may join and existing members may leave, and routes may also change due to dynamic status changes at intermediate switches and links. To be able to adjust resource reservations accordingly, and in a way transparent to end applications, RSVP keeps *soft-state* at intermediate switches and leaves the responsibility of maintaining the reservation to end users. The term soft-state was first used by Clark in [3] and, in our context, refers to state maintained at network switches which, when lost, will be automatically reinstated by RSVP soon thereafter. Thus, soft-state is appropriate in our context where frequent routing changes and occasional service outages would render a more brittle (i.e., less self-stabilizing) state to become, and perhaps remain, obsolete or incorrect.

More specifically, RSVP distinguishes two kinds of state information at each intermediate switch, *path* state and *reservation* state. Each data source periodically sends a *Path* message that establishes or updates the path state, and each receiver periodically sends a *Reservation* message that establishes or updates the reservation state (which is attached to the path state).

Path messages are forwarded using the switches' existing routing table; in other words the routing decision is made by the network's routing protocol, not by RSVP. Each path message carries a flowspec given by the data source, as well as an F-flag indicating if the application wishes to allow filtered reservations. In processing each path message, the switch updates its path state that contains information about (1) the incoming link upstream to the source, and (2) the outgoing links downstream from that source to the receivers in the group (as indicated by the multicast routing table). In addition, if the F-flag in the path message is on, the switch also keeps the information about the source and the previous hop upstream to reach the source; this information allows the switch to accommodate any style of reservation. If the F-flag is off, the switch will not maintain any information about the specific source of the path message except adding its incoming link to the path state, thus minimizing the state that must be kept at the switch. Consequently, only no-filter style reservations can be made for data streams from such sources. As we will show later in an example, not maintaining per source information can, in some topologies, result in over-reserving resources over certain links.

Each reservation message carries a flowspec, a reservation style, and a packet filter if the reservation uses a filtered style (either fixed or dynamic). In processing each reservation message, the switch updates its reservation state that contains information for the outgoing link the message came from by recording (1) the amount of resources reserved, (2) the source filter for the reserved resource, (3) the reservation style, and if the style is dynamic-filter, (4) the reserver (the sender of this reservation message, which is one of the receivers of this multicast group). We see that the only time we need to keep per receiver information in the reservation table is when the reservations involve dynamic filters; when all reservations are either no-filter or fixed-filter we can assign the reservation to the multicast group as a whole and then only keep track of the total resources reserved on each downstream link.

Reservation messages are forwarded back towards the sources by reversing the paths of path messages. In fact, the path information is maintained solely to do this reverse path forwarding of the reservation messages. More specifically, reservation messages of the no-filter style are forwarded to all incoming links to the multicast group, and those of filtered styles are forwarded to the previous hops of the sources that are listed in the filters.

Both path messages and reservation messages carry a timeout value that is used by intermediate switches to set corresponding timers; the timers get reset whenever new messages are received. Whenever a timer expires, the corresponding state will be deleted. This timeout-driven deletion prevents resources from being orphaned when a receiver fails to send an explicit tear-down message or the underlying route changes. It is also the only way to release the resources of no-filter or fixed-filter reservations; in these cases, the switch cannot determine if the reservation is being shared by multiple receivers, and so can only delete the reservation when it times out. It is the responsibility of both senders and receivers to maintain the proper reservation state inside the network by periodically refreshing the path and reservation state.

When a route or membership changes, the routing protocol running underneath RSVP will forward future path messages along the new route(s) and reach new members. As a result, the path state at switches will be updated, causing future reservation messages to traverse the new routes or new route segments. Reservations along old routes, or along routes to inactive senders or receivers will time out automatically. Because path and reservation messages are sent periodically, the protocol will tolerate occasional corruption or loss of a few messages. This soft-state approach adds both adaptivity and robustness to RSVP.

The advantages of the soft-state approach, however, do not come for free; the periodic refreshing messages add overhead to the protocol operation. We next discuss how RSVP controls protocol overhead.

3.5 Protocol Overhead Control

The RSVP overhead is determined by three factors: the number of RSVP messages sent, the size of these RSVP messages, and the refresh frequencies of both path and reservation messages. As we describe in more detail in Section 4, RSVP merges path and reservation messages as they traverse the network. The merging of path messages means that, in general, each link carries no more than a single path message in each direction during each path refresh period. Similarly, the merging of reservation messages means that each link carries no more than a single reservation message in each direction during each reservation refresh period. The maximum size of both the path and reservation messages on a particular link is proportional to the number of data sources upstream.

RSVP controls the third overhead factor, the refresh frequencies, by tuning the timeout values carried in path and reservation messages. The larger the timeout value, the less frequent the refresh messages have to be sent. There exists, however, a tradeoff between the overhead one is willing to tolerate and RSVP's responsiveness in adapting to dynamic changes. For instance, reservation messages are forwarded according to the path state maintained at intermediate switches, which in turn gets synchronized with the routing protocol state every time a path message is processed. When a route changes, reservations along the new route (or new route segments) will not be established until a new path message has been sent (causing the path state to be updated), and a new reservation message has been sent along the new route.

Our current RSVP implementation uses static timer values which are chosen on the basis of engineering judgment; in the future we will investigate adaptive timeout algorithms to optimally adjust the timer values according to observed dynamics in route and membership changes, as well as the loss probability of RSVP messages.

3.6 Modularity

In the context of a real-time, multicast application, RSVP interfaces to three other components in the architecture: (1) the flowspec, which is handed to RSVP by an application, or some session control protocol on behalf of the application, when invoking RSVP; (2) the network routing protocol, which forwards path messages towards all the receivers, causing RSVP path state to be established at intermediate switch nodes; and (3) the network admission control, which makes an acceptance decision based on the flowspec carried in the reservation messages.

We list modularity as one of RSVP's design goals because we would like to make RSVP as independent from the other components as possible. We have attempted to make few assumptions about these other components, and those assumptions that we have made are described explicitly.

We make no assumptions about the flowspec to be carried by RSVP. RSVP treats the flowspec as a number of uninterpreted bytes of data that need to be exchanged among only the applications and the network admission control algorithm. We assume that the admission control algorithm operates by having an RSVP reservation packet containing a flowspec pass through the switches along the delivery path for that flow (but obviously in the reverse direction), with each switch returning an *admit* or *reject* signal; the resource reservation is established only if all switches along the path admit the flow. We also assume that the packet scheduling algorithm can change packet filters without needing to establish a new reservation.

The only assumptions that we make about the underlying routing protocol(s) are that it provides both unicast and multicast routing, and that a sender to a multicast group can reach all group members under normal network conditions; obviously, in the case of a network partition no routing protocol can guarantee this reachability. We do not assume that a sender to a multicast group is necessarily a member of the group, nor do we assume that the route from a sender to a receiver is the same as the route from the receiver to the sender.

4 RSVP Operation Overview

RSVP, and indeed any reservation protocol, is a vehicle for establishing and maintaining state in switches along the paths that each flow's data packets will travel. Because reservation messages are initiated by each receiver, RSVP must make sure that the reservation messages from a receiver follow exactly the reverse routes of the data streams from all the sources (that the receiver is interested in). In other words, RSVP must establish a *sink tree* from each receiver to all the sources to forward reservation messages.

The sink tree for each receiver is formed by tracing, in the reverse direction, the paths (as defined by the multicast routing protocol) from the receiver to each of the sources (see Figure 4). Periodic path messages are forwarded along the routing trees provided by the routing protocol, and reservation refresh messages are forwarded along the sink trees to maintain current reservation state. However, a reservation message propagates only as far as the closest point on the sink tree where a reservation level greater than or equal to the reservation level being requested has already been made.

Each switch uses the path states to maintain, for each multicast group, a table of incoming and outgoing interfaces. Each incoming interface keeps the information about the flowspecs it has forwarded upstream (which is needed in merging reservation requests from multiple downstream links). For each outgoing link, there is a list of senders; associated with each sender in this list is the previous hop address from which data from that sender arrives at the current switch. There is also a set of reservations. Generally speaking, each reservation consists of a reserver, a filter, and the amount of resources reserved. For no-filter reservations, the first two fields are not needed; and for fixed-filter reservations, the first field is not needed.

We now review the process of creating and maintaining reservations in more detail. Before or when each data source starts transmitting, it sends a path message which contains the flowspec provided by the data source. When a switch receives a path message, it first checks to see if it already has the path state for the named target (which can be either a single host or a multicast group, plus the destination port number); if not it creates path state for that target. The switch then obtains the outgoing interface(s) of the path message from the routing protocol in use, and updates its table of incoming and outgoing links accordingly; the source address (and port number in the Internet context) carried in the path message will also be recorded if the path message indicates that the application may require a filtered reservation. This path message is forwarded immediately only if it is from a new source or indicates a change in routes. The switch can detect a change in routes by checking to see if the outgoing interfaces indicated by the routing protocol's routing table are different than the outgoing links maintained in the path state. Otherwise, the switch discards the incoming path message and instead periodically sends its own path messages which contain the path information carried in all the path messages that it has received so far.

When a receiver receives a path message from a source for whose data it would like to create a reservation, the receiver sends a reservation message using the (possibly modified) flowspec that came in the incoming path message. As described earlier, the reservation message will be guided along the reverse route of the path messages to reach the data source(s). Along the way if any switch rejects the reservation, an RSVP reject message will be sent back to the receiver and the reservation message discarded; otherwise if the reservation message requires a new reservation to be made, it will propagate as far as the closest point along the way to the sender(s) where a reservation level equal or greater than that being requested has been made.

Once the reservation is established, the receiver periodically sends reservation refresh messages (which are identical in format to the original request). As the reservation requests are forwarded along the sink trees, the switches merge the requests for the same multicast group by pruning those that carry a request for reserving a smaller, or equal, amount of resources than some previous request. As an example, let us assume that H1 in Figure 4 is a video source and that H4 has reserved enough bandwidth to receive the full video data stream while H5 wants to receive only low resolution video data. In this case, when the reservation request from H5 reaches S4, S4 will make the requested reservation over the link from S4 to H5, and then S4 will drop the request (i.e., not forward it upstream) because sufficient resources have been reserved already by H4's request.

When a sender (receiver) wishes to terminate the connection, the sender (receiver) sends out a path (reservation) teardown message to release the path state or reserved resources. There is no retransmission timer for this teardown message. In cases where the teardown message is lost, the intermediate nodes will eventually time out the corresponding state. As we noted above, no-filter or fixed-filter reservations cannot be explicitly torn down because the switches do not maintain sufficient state.

5 Example

To illustrate in more detail how RSVP works, we consider a simple network configuration. There are 5 hosts connected by 7 point-to-point links and 3 switches (we assume that for the links connecting them directly to a switch, the hosts act as switches in terms of reserving resources). To simplify the description, in the following examples we assume adequate network resources exist for all reservation requests. Furthermore, the example involves only a single multicast group, so we do not discuss the addressing used to distinguish reservations made for one multicast group from reservations made for other multicast groups.

We describe the cases of no-filter and filtered reservations separately; we start with the simpler case, no-filter reservations, and then discuss the case of filtered reservations.

5.1 No-filter Reservations

Let us consider an audio conference to be held among 5 participants, one at each of the 5 hosts depicted in Figure 5 (therefore each host behaves both as a source and a receiver at the same time). We assume that (1) the routing protocol has built a multicast routing tree so that each sender can reach all the receivers; (2) each switch has received RSVP path messages with the F-flag *off* from all the sources (therefore the switches do not record source information) and stored complete path state, as below (although in a real application sources may start at different times hence the path state would be built up over time); and (3) no reservations have been made yet.

	S1	S2	S3
Incoming-links	L1, L2, L6	L5, L6, L7	L3, L4, L7
Outgoing-links	L1, L2, L6	L5, L6, L7	L3, L4, L7

We now describe how reservations are created. H1 wants to receive data from all other senders to the multicast group, but only wants enough bandwidth reserved to carry one audio stream; thus, it sends a reservation message `R1(B, no-filter)` to S1, where B is the amount of bandwidth needed to forward one audio stream. When S1 receives `R1(B, no-filter)`, it first reserves resources over L1 (in the direction from S1 towards H1), then attaches the following reservation state to the path state to indicate the amount of the reservation made over L1:

	S1		
Incoming-links	L1	L2	L6
Outgoing-links	L1(B)	L2	L6

Finally, S1 forwards `R1(B, no-filter)` over all incoming-links, in this case L2 and L6. Note that the switch never forwards any RSVP message over the link the message came from.

The copy of `R1(B, no-filter)` that was sent along L6 reaches S2, which reserves B over L6 and forwards the message to links 5 and 7. When the copy of `R1(B, no-filter)` that was sent along L7 reaches S3, that switch reserves B over L7 and then forwards `R1(B, no-filter)` over links 3 and 4.

When H2 wants to create a reservation, it sends a reservation message, `R2(B, no-filter)`, to S1. Upon receipt of `R2(B, no-filter)`, S1 first reserves B over L2, so the path state then becomes:

	S1		
Incoming-links	L1	L2	L6
Outgoing-links	L1(B)	L2(B)	L6

S1 then forwards `R2(B, no-filter)` over L1 only, because it has forward an identical request over L6 previously.

After all the receiving hosts have sent RSVP reservation messages, an amount B of resources have been reserved over each of the 7 links in each of the two directions.

Before leaving this example of no-filter reservation, let us consider the tradeoff between keeping extra state information and the possibility of over-reserving resources on certain links as we mentioned earlier. In the above example we had assumed that all the path messages had the F-flag off, therefore there is no per source information kept at the switches. As a result, if each of the receivers had requested an amount 2B of bandwidth (i.e., an amount enough to carry two full audio streams), then an amount 2B would be reserved on every link even though on link L1 (and similarly on links L2, L3, L4, and L5) in the direction away from H1 we need only reserve an amount B since there is only a single source upstream on the link. In general, a no-filter reservation should indicate how much should be reserved as a function of the number of sources upstream; in this example it would be B units per upstream source. Unfortunately, one cannot know the number of sources upstream without keeping a list of the sources. Had the F-flag been set in all the path messages, the switches would have kept track of individual sources and by paying this extra cost in increased state, only the required amount of resources would have been reserved along all the links.

Although not maintaining per source information can lead to an over-reserving of resources over some of the network links, as the above example showed, in those applications where there are many data sources, but few resources are needed for each source (such as in a data-gathering application with many sensors), one may still choose to reduce the switch state at the possible expense of over-reserving resources over some links.

5.2 Filtered Reservations

Now consider the case where H2, H3, H4, and H5 are receivers (i.e., members of the multicast group), and H1, H4, and H5 are sources. All path message have the F-flag set, so each switch needs to keep a list of sources associated with their previous hops. Assume that S1 has received path messages from all of the sources, but that no reservations have yet been made. Thus, S1's path state contains the following entry:

<table>
<tr><th></th><th colspan="2">S1</th></tr>
<tr><td>Outgoing-links</td><td>L2(src:H1,H1 | H4,S2 | H5,S2)</td><td>L6(src:H1,H1)</td></tr>
</table>

The notation `L2(src:H1,H1 | H4,S2 | H5,S2)` indicates that data from sources H1, H4, and H5 are sent out along outgoing link L2; for each source, `H1`, `S2`, and `S2` are the previous hop addresses from which data from that source arrives, respectively. H1 is not a receiver, so L1 is not among the outgoing links of S1.

Now assume that H2 sends the following reservation message, denoted `R2(B, H4)`. That is, H2 wants to receive packets only from source H4, and is reserving an amount B that is sufficient for one source. The reservation message `R2(B, H4)` reaches S1 via the L2 interface. S1 finds that H4 is indeed one of the sources it has heard, and that the packets from H4 come from S2. S1 reserves bandwidth B over L2, and forwards `R2(B, H4)` over L6 to S2.

S2's path state contains the following entries:

<table>
<tr><th></th><th colspan="3">S2</th></tr>
<tr><td>Outgoing-links</td><td>L5(src:H1,S1 | H4,S3)</td><td>L6(src:H4,S3 | H5,H5</td><td>L7(src:H1,S1 | H5,H5)</td></tr>
</table>

When S2 receives `R2(B, H4)`, it reserves B over L6, and then forwards the message `R2(B, H4)` to S3 (which is the previous hop towards H4).

S3's path state contains the following entries:

<table>
<tr><th></th><th colspan="3">S3</th></tr>
<tr><td>Outgoing-links</td><td>L3(src:H1,S2 | H4,H4 | H5,S2)</td><td>L4(src:H1,S2 | H5,S2</td><td>L7(src:H4,H4)</td></tr>
</table>

Upon receiving `R2(B, H4)`, S3 reserves B over L7, and forwards the message to H4. When the message reaches H4, a pipe of B has been reserved from H4 to H2. This describes the reservation events surrounding the reservation request `R2(B, H4)`.

Suppose that sometime afterwards, H5 sends the reservation message `R5(2B, *)`, where `*` indicates a request for dynamic-filter reservation. When S2 receives this reservation message `R5(2B, *)`, it reserves 2B over L5 (at least 2 sources can go that direction) for H5; and forwards the reservation message `R5(2B, *)` over L6 and L7.

When S1 receives `R5(2B, *)`, it finds out that there is only one source going out L6. It therefore reserves an amount B over L6 for R5 and then passes the reservation request on to H1.

When S3 receives `R5(2B, *)`, it finds out that there is only one source going out L7, and has a fixed-filter reservation already. S3 does not reserve any more, nor does it further forward the request to L4.

Suppose now that at some point H4 decides to terminate both receiving and sending, and does not transmit any teardown messages. Since H4 will no longer be sending path or reservation refreshes, all H4 related state will time out, resulting in the following outgoing-link entries in the various switches:

S1	`L2(src:H1,H1 | H5,S2)`		`L6(src:H1,H1)`
S2	`L5(src:H1,S1)`	`L6(src:H5,H5)`	`L7(src:H1,S1 | H5,H5)`
S3	`L3(src:H1,S2 | H5,S2)`		

S1 stops forwarding `R2(B, H4)` from H2 and returns an RSVP error message to H2. S2 forwards future `R5(2B, *)` reservation refreshes to the L6 direction only since there are no more sources in L7 direction.

For the sake of simplicity, in the above example we have assumed that each of the data streams requires the same amount of bandwidth to forward. RSVP is designed to handle the case where each source may demand different amount of resources, and each receiver may receive only a subset of the data from each source. In fixed-filter reservations, this requires that each source filter must be associated with a specific amount of resources. In dynamic-filter reservations, the receiver must either receive the same amount of data when "switching channels", or it must specify a specific amount of resources for each of the sources in its current filter, and the sum of its total incoming data volume does not change over the lifetime of the reservation.

6 Implementation Status

This article is intended to illustrate, at a general level, how RSVP works. There are many details that, for the sake of brevity and clarity, we have not presented. In particular, we have not described with any specificity the merging algorithm. However, we have verified this design in a packet-level, interactive simulator, where all such details have been tested.

The simulator used is written by one of us (LZ), and has been used in several previous simulation studies ([4, 10, 18]). The simulator provides modules that imitate the actual behavior of common network components, such as hosts, links, IP routers, and protocols such as IP, TCP, and UDP. We verified RSVP design by implementing the protocol in the simulator and then observing, step by step, how the protocol handles various dynamic events, such as new senders/receivers joining a multicast group, or existing members leaving. Indeed, the design of most protocol details emerged from an iterative process of simulation and redesign.

Using the simulator code as a starting point, the protocol has been implemented by Sugih Jamin (USC) for experimentation on Dartnet, which is a cross-country T1 network testbed sponsored by ARPA linking roughly a dozen academic and industrial research institutions. Preliminary tests have been performed on this implementation, but no systematic performance studies have been done as yet.

7 Related Work

In the course of exploring network algorithms that deliver quality of service guarantees, there have been several proposals and prototype implementations of network resource reservation algorithms over the last few years (see, for example, [6], [2]). However, almost all of these prototypes deal exclusively with unicast reservations.

The Stream Protocol, ST [7], was a pioneering work in multicast reservation protocol design. ST was designed specifically to support voice conferencing and was capable of making both unicast and multicast resource reservations. At the time ST was proposed, there was no work on sophisticated multicast routing, so ST would make resource reservations over a single, duplex distribution tree which was created by blending the paths from unicast routing; this was done with the assumptions that the routes were reversible and the application data traffic would travel in both directions. However ST requires a centralized Access Controller to coordinate among all the participants and to manage the tree establishment.

The successor to ST, ST-II [17], continues to create its own multicast trees by blending the paths from unicast routing; however, ST-II establishes multiple simplex reservations to eliminate the Access Controller. Each data source makes a resource reservation along a multicast tree that is rooted at the source and reaches out to all the receivers; the reservation made along the tree uses a single flowspec, therefore ST-II cannot accommodate heterogeneous receivers. Because each data source makes its reservation independently, a single pipe is reserved from every source to every receiver in the same multicast application group. An analysis of ST-II implementation and design issues is provided in [15].

Thus, neither ST nor ST-II provides a robust and efficient solution to the multipoint-to-multipoint resource reservation problem; they share several of the limitations of the strawman proposal described earlier. The RSVP design effort was initiated to fill this vacuum. Recently, however, there have been other proposals to fill this need. Pasquale et al. have proposed a dissemination-oriented approach in their work on multimedia multicast channels [16]. They share with us the viewpoint that, in order to efficiently support heterogeneous receivers, each receiver must be able to specify a stream filter for the subset of the data it is interested in receiving, and furthermore that, in order to not waste network resources, the filters from all the receivers should be propagated towards the sender so that the subset of the data that no one is interested would be stopped at the earliest point along the source propagation tree. However, they only considered single source applications (such as cable TV), as opposed to RSVP's functionality of supporting multipoint-to-multipoint applications, and they have mainly focused on the programming interface to applications, as opposed to our interest in designing a protocol that reserves resources inside the network and adjusts the reservation to dynamic environmental changes.

8 Unresolved Issues

While RSVP has been simulated and tested to some extent, we fully expect that further incremental design changes will be made as we gain experience with RSVP, both on DARTnet and also through further simulation. Besides these incremental changes, however, there are several larger design issues that remain unresolved. These issues are:

- RSVP was designed with minimal expections of routing. Path messages are used to essentially invert the routing tables, a function that routing could easily provide if it were so designed. If we were to design new routing algorithms, what routing support would we include to support resource reservation algorithms?
- In this design we have associated filters with resource reservations. In fact, filters could be applied to flows even without reserved resources. Furthermore, there are filter styles besides the ones described here that might be useful. For instance, as has been proposed by Jacobson [9], for remote lectures with several speakers at separate sites one might want a dynamic filtered reservation where the filter is the same for each receiver; this feature would allow the audience to switch (in unison) to different speakers with only one set of resources reserved. Thus, one unresolved issue is defining the general service model and interfaces for such filters, where these definitions are not specifically tied to the presence of resource reservations.
- Our current simulations and tests deal only with reasonably small networks and small multicast groups. We do not yet understand how RSVP performs when the size of the multicast groups gets very large. Can one use caching strategies to avoid the router state explosion when S (the number of senders) and/or R (the number of receivers) gets very large? This issue is particularly relevant to the case of cable TV, where every home would want a dynamic reservation, but the switches obviously would not want to keep individual reservation state for each home.

9 Summary

RSVP's architecture is unique in that: (1) it provides receiver-initiated reservations to accommodate heterogeneity among receivers as well as dynamic membership changes; (2) it separates the filter from the reservation, thus allowing channel changing behavior; (3) it supports a dynamic and robust multipoint-to-multipoint communication model by taking a *soft-state* approach in maintaining resource reservations; and (4) it decouples the reservation and routing functions and thus can run on top of, and take advantage of, any multicast routing protocols.

We have verified the first RSVP design, as described above, by detailed simulation and a preliminary implementation. Much testing remains to be done in the context of larger scale simulations, as well as in real prototype networks such as DARTnet.

10 Acknowledgments

We would like to gratefully acknowledge useful conversations with Bob Braden, David Clark, Ron Frederick, Shai Herzog, Sugih Jamin, and Danny Mitzel.

References

[1] Ballardie, A., Tsuchiya, P., and Crowcroft, J., *Core Based Trees (CBT)*, Internet Draft, November, 1992.

[2] Cidon, I., Segall, A., *Fast Connection Establishment in High Speed Networks*, **in the Proceedings of ACM SIGCOMM '90**, September, 1990.

[3] Clark, D. D. *The Design Philosophy of the DARPA Internet Protocols*, **in the Proceedings of ACM SIGCOMM '88**, August, 1988.

[4] Clark, D. D., Shenker, S., and Zhang, L. *Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism*, **in the Proceedings of ACM SIGCOMM '92**, August, 1992.

[5] Deering, S., *Multicast Routing in a Datagram Internetwork*, **Tech. Report No. STAN-CS-92-1415, Stanford University**, December, 1991.

[6] Ferrari, D., Banerjea, A., and Zhang, H., *Network Support for Multimedia: A Discussion of the Tenet Approach*, Technical Report TR-92-072, Computer Science Division, University of California at Berkeley, November 1992.

[7] Forgie, J., *ST – A Proposed Internet Stream Protocol*, Internet Experimental Notes IEN-119, September 1979.

[8] Golestani, S. J., *Duration-Limited Statistical Multiplexing of Delay Sensitive Traffic in Packet Networks*, In **Proceedings of INFOCOM '91**, 1991.

[9] Jacobson, V., private communication

[10] Jamin, S., Shenker, S., Zhang, L., and Clark, D., *Admission Control Algorithm for Predictive Real-Time Service*, **Proceedings of 3rd International Workshop on Network and Operating System Support for Digital Audio and Video**, November, 1992.

[11] Kalmanek, C., Kanakia, H., and Keshav, S., *Rate Controlled Servers for Very High-Speed Networks*, In **Proceedings of GlobeCom '90**, pp 300.3.1-300.3.9, 1990.

[12] J. Hyman, A. Lazar, and G. Pacifici. *Real-Time Scheduling with Quality of Service Constraints*, In **IEEE JSAC**, Vol. 9, No. 9, pp 1052-1063, September 1991.

[13] Hyman, J. M., Lazar, A. A., Pacifici, G.: Joint Scheduling and Admission Control for ATS-based Switching Nodes: Proc. ACM SIGCOMM '92, August, 1992.

[14] Partridge, C., *A Proposed Flow Specification*, **Internet RFC-1363**, July, 1992.

[15] C. Partridge and S. Pink. *An Implementation of the Revised Internet Stream Protocol (ST-2)*, In **Internetworking: Research and Experience**, Vol. 3, No. 1, pp 27-54, March 1992.

[16] Pasquale, J., Polyzos, G., Anderson, E., and Kompella, V., *The Multimedia Multicast Channel*, **Proceedings of 3rd International Workshop on Network and Operating System Support for Digital Audio and Video**, November, 1992.

[17] Topolcic, C., *Experimental Internet Stream Protocol: Version 2 (ST-II)*, **Internet RFC 1190**, October, 1990.

[18] Zhang, L., *A New Architecture for Packet Switching Network Protocols*, In **Technical Report TR-455, Laboratory for Computer Science, Massachusetts Institute of Technology**, 1989.

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

Figure 1: The Seven Design Goals of RSVP.

1. Receiver-Initiated Reservation
2. Separating Reservation from Packet Filtering
3. Providing Different Reservation Styles
4. Maintaining "Soft-State" in the Network
5. Protocol Overhead Control
6. Modularity

Figure 2: The Six Design Principles of RSVP.

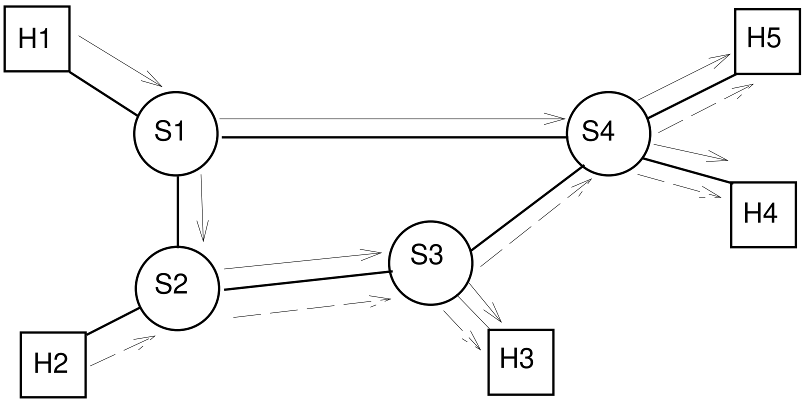

Figure 3: A simple network topology with the multicast routing trees depicted. H1 and H2 are data sources, and H3, H4, and H5 are receivers. The solid lines depict the routing tree of H1 and the dotted lines the routing tree of H2. In general, the set of sources and the set of receivers may overlap partially or completely; here, for the sake of clarity, we consider the case where they are disjoint.

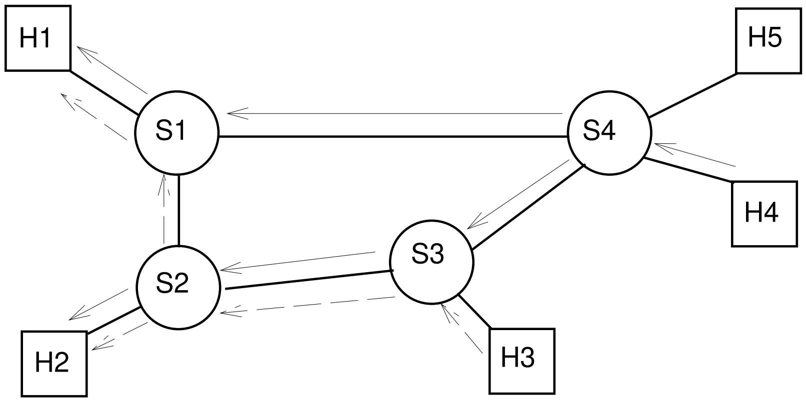

Figure 4: A simple network topology with the sink trees depicted. H1 and H2 are data sources, and H3, H4, and H5 are receivers (sinks). The dotted lines depict the sink tree of H3 and the solid lines the sink tree of H4. For clarity the sink tree of H5 is omitted.

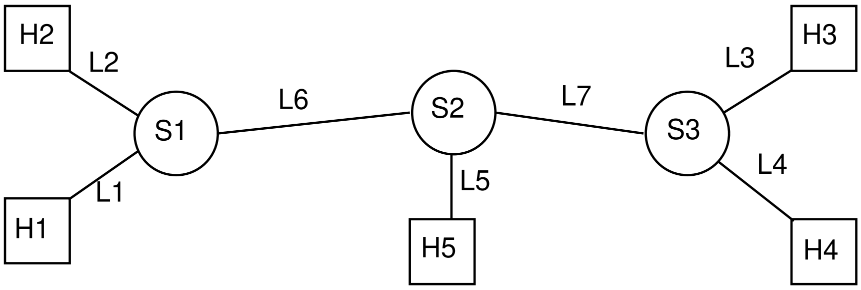

Figure 5: Network Topology.