RSVP: A New Resource ReSerVation Protocol

Daniel Zappala
daniel_zappala@byu.edu

Stephen Deering

See next page for additional authors

Follow this and additional works at: https://scholarsarchive.byu.edu/facpub

Part of the Computer Sciences Commons

Original Publication Citation

BYU ScholarsArchive Citation
Zappala, Daniel; Deering, Stephen; Estrin, Deborah; Shenker, Scott; and Zhang, Lixia, "RSVP: A New Resource ReSerVation Protocol" (1993). All Faculty Publications. 706.
https://scholarsarchive.byu.edu/facpub/706

This Peer-Reviewed Article is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Faculty Publications by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
RSVP: A NEW RESOURCE RESERVATION PROTOCOL

Lixia Zhang, Stephen Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala

Originally published in
IEEE Network Magazine
September 1993 — Volume 7, Number 5

AUTHOR’S INTRODUCTION

The origin of the RSVP protocol can be traced back to 1991, when a team of network researchers, including myself, started playing with a number of packet scheduling algorithms on the DARTNET (DARPA Testbed Network), a network testbed made of open source, workstation-based routers. Because scheduling algorithms simply shuffle packet processing orders according to some established rates or priorities for different data flows, to test a scheduling algorithm requires setting up the appropriate control state at each router along the data flow paths. I was challenged to design a set-up protocol that could support both unicast and many-to-many multicast applications. That effort led to the birth of RSVP.

As a signaling protocol designed specifically to run over IP, RSVP distinguishes itself from previous signaling protocols in several fundamental ways. The most profound ones include a soft-state approach, two-way signaling message exchanges, receiver-based resource reservation, and being independent from all other related components in a QOS support architecture, such as flow-specification, admission control, scheduling algorithm, and routing. As stated in the article, “RSVP is primarily a vehicle used by applications to communicate their requirements to the network in a robust and efficient way, independent of the specific requirements.”

It has been more than 10 years since the original idea was first conceived. Over this time period many people contributed to the effort that has evolved RSVP from a lab toy to a Proposed Internet Standard Protocol. Other more recent protocol developments, such as MPLS (Multi-Protocol Label Switching), VPN (Virtual Private Network), and OTN (Optical Transport Network), to name a few, have adopted or considered RSVP for their own signaling use. I was stunned by RSVP’s rapid adoption and development of usage. The protocol has moved on with a life of its own. I have learned many lessons from observing which features in the original design worked and which didn’t. Among these lessons, I noticed that the proposal of supporting flexible resource reservations by individual users is yet to prove useful, and that the decision to make RSVP a generic messenger, which simply carries “a bag of bits” to pass to routers along the way, has proven to be a right one, which promoted the adoption of RSVP for various purposes other than QOS support.

The effort that started RSVP design is but the first step in developing signaling protocols for the Internet. Although the debate on which kinds of QOS support the Internet would need continues, various signaling needs demand a generic signaling protocol. Independent from whether RSVP would be the last one to fulfill that important role, I believe the basic principles and lessons we have gained from RSVP development will extend beyond the protocol itself into new protocol designs for the future Internet.
The current Internet architecture, embodied in the Internet Protocol (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. 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, they are often multi-point-to-multipoint, with several senders and receivers of data. Multipoint-to-multipoint communication occurs, for example, in multiparty conferences where each participant is both a sender and a receiver of data, and also in remote learning applications, although in the latter 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 fundamental differences exist between the 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 a source can tell the network about the traffic characteristics of its flow and, in turn, the network can 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 both the characteristics of the traffic stream sent by the source, and 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 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 the literature [1, 2].

**Routing:** The network must decide how to transport packets from the source to the receiver of the flow (or receivers of the flow, in the case of multicast). Thus, the second component of the architecture is a routing protocol that provides quality unicast and multicast paths. There are many approaches to unicast routing, and several different approaches to multicast routing exist as well [2-4]. 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:** For the network to deliver a quantitatively specified quality of service (e.g., a bound on delay) to a particular flow, it is usually necessary 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. Two approaches to resource reservation are described elsewhere [2, 5]; 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 that determines which reservation requests to grant and which to deny, thereby maintaining the network load at an appropriate level. Two such admission control algorithms are described in the literature [6, 7].

**Packet Scheduling:** After every packet transmission, a network switch must decide whether or not to transmit the next packet, and which is next. These decisions are controlled by the packet scheduling algorithm, which lies at the heart of any network architecture because it determines the qualities of service the network can provide. There are many proposed packet scheduling algorithms. A few examples are cited here [8-12].

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 [2], 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: 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 (i.e., 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 is organized as follows. We first list our design goals, and then discuss the basic design principles used to meet these goals. A more detailed description of the protocol opera-
The strawman proposal here 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.

RSVP DESIGN GOALS

In the traditional point-to-point case, one obvious reservation paradigm would have the sender transmit a reservation request toward 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 they are not addressed by the strawman proposal of straightforwardly extending the basic paradigm. In the process, we identify the seven goals that guided the design of RSVP.

In a wide-area internetwork such as the Internet, receivers and paths to reach 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. Certain receivers 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 to reach 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. Although appropriate for some applications, in other cases this duplication can lead to a significant wasting of resources. For example, in an audio conference with several people, usually only one person, or at most a few people, talk 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 is adequate to reserve only enough network resources to handle a few simultaneous audio channels. Our third design goal for RSVP is to allow end users to specify their application needs, so the aggregate resources reserved for a multicast group can more accurately reflect the resources actually required by the group.

Furthermore, in a multiparty conference a receiver may only wish to (or 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; it would be expensive and complicated to replicate such functions 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 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 architectural components listed in the first section of this article. 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 [4], the recently proposed core-based tree (CBT) multicast routing [3], or some future routing protocols. This design goal makes RSVP deployable in many contexts. For optimally efficient routing decisions, however, routing selection and resource reservation should be integrated, so the choice of route can depend on the quality of service requested, and 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.

In summary, we have identified seven important design goals (see box this page). RSVP is primarily a vehicle used by applications to communicate their requirements to the network in a robust and efficient way, independent of the specific requirements. RSVP delivers resource reservation requests to the relevant switches but plays no other role in providing network services. Thus, RSVP communicates requirements for a wide range of network services but does not directly provide them. 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 advanced reservations (reservations made for a future time) and preemptable reservations (reservations that a receiver is willing to have preempted). RSVP is capable of supporting the delivery of these and other services, whenever these network services rely only on the state being established at the individual switches along the paths determined by the routing algorithm. Thus, although we described RSVP as a resource reservation protocol, it can be seen more generally as a “switch-state establishment” protocol.

**BASIC DESIGN PRINCIPLES**

To achieve the seven design goals, we used six basic design principles (see box this page). These principles are now described.
A separate function, called a packet filter, selects those packets that can use the resources; it is set by the reserving entity. 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.

**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 that 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 are reserved for whom. Here, "whom" does not refer to "which packets" can use the reserved resources; rather, it refers to "which entity" controls the resources.

A separate function, called a packet filter, selects those packets that can use the resources; it is set by the reserving entity. Moreover, it can be changed without changing the amount of reserved resources. 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 enables us to offer several different reservation styles, which we now describe.

**PROVIDING DIFFERENT RESERVATION STYLES**

As we discussed briefly above, the service requirements of multicast applications dictate how the reservation requests from individual receivers should be aggregated inside the network. For example, 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 is 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 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 no filter is used, then any packets destined for that multicast group may use the reserved resources. (Although some enforcement mechanism is needed to ensure 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 whoever 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 is identified by the source host address plus source port number. We only refer to the source host address here.) Only the packets from the specified sources can use the reserved resources. Filtered reservations are 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 allows a receiver to receive data only from the sources listed in the original reservation request, for the duration of the reservation. A "dynamic-filter" reservation allows a receiver to change its filter to different sources over time.

To illustrate how intermediate nodes use these reservation styles to aggregate reservation requests, consider the case of several receivers in the same multicast group making 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) is reserved for each source even when there are multiple requests from different sources. 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. The intermediate nodes cannot aggregate this style of reservation because the reservations may be made for the maximum number of incoming streams it can handle at once and the network reserves 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 to aggregate reservations 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.

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, new members may join, existing members may leave, and routes may change due to dynamic status changes at intermediate switches and links. To be able to adjust resource reservations accordingly, 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 [13]. In our context, it refers to a 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 membership 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, at each intermediate switch, RSVP distinguishes between state information of two kinds: 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 containing information about 1) the incoming link upstream to the source, and 2) the outgoing link 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 does not maintain information about the specific source of the path message except for adding its incoming link to the path state; the state kept at the switch is thereby minimized. Consequently, only no-filter style reservations can be made for data streams from such sources.

As we 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 (if the reservation uses a fixed or dynamic filtered style) a packet filter. In processing each reservation message, the switch updates its reservation state (which 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 4) if the style is dynamic-filtered, the reserver (who is the sender of this reservation message, and 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 toward the sources by reversing the paths of path messages. In fact, the path information is maintained solely for this reverse-path forwarding of 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 used by intermediate switches to set corresponding timers; the timers get reset whenever new messages are received. Whenever a timer expires, the corresponding state is deleted. This timeout-driven deletion...
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 frequently 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) are not established until a new path message is sent (causing the path state to be updated), and a new reservation message is sent along the new route.

Our current RSVP implementation uses static timer values 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 routes and membership changes, and the loss probability of RSVP messages.

**Modularity**

In the context of real-time, multicast applications, RSVP interfaces to three other components:

- The flowspec, which is handed to RSVP by an application or some session-control protocol on behalf of the application, when invoking RSVP.
- The network routing protocol, which forwards path messages toward all the receivers, causing the RSVP path state to be established at intermediate switch nodes.
- 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 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 about the underlying routing protocol(s) are that it provides both uni-

---

**Figure 1.** A simple network topology with the multicast routing trees. H1 and H2 are data sources, and H3, H4, and H5 are receivers. The solid lines depict the routing tree of H1; the dotted lines depict the routing tree of H2. In general, the set of sources and the set of receivers may overlap partially or completely. For the sake of clarity, here they are disjoint.
cast 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.

**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 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 stream 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 the paths defined by the multicast routing protocol — in the reverse direction — from the receiver to each of the sources (Figs. 1 and 2). 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. 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 a table of incoming and outgoing interfaces for each multicast group. Each incoming interface keeps the information about the flowspecs it has forwarded upstream. (This information is needed in merging reservation requests from multiple downstream links.) For each outgoing link, there is a list of senders; associated with each sender is the address and port number; if not, it creates a reservation state. A reservation message propagates along the sink tree where a reservation level greater than or equal to the reservation level being requested has already been made.

The sink tree for each receiver is formed by tracing the paths defined by the multicast routing protocol — in the reverse direction — from the receiver to each of the sources (Figs. 1 and 2). 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. 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 a table of incoming and outgoing interfaces for each multicast group. Each incoming interface keeps the information about the flowspecs it has forwarded upstream. (This information is needed in merging reservation requests from multiple downstream links.) For each outgoing link, there is a list of senders; associated with each sender is the address and port number; if not, it creates a reservation state. 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.

When a receiver receives a path message from a source for whose data it would like to create a reservation, it sends a path message containing the flowspec that came in the incoming path message. As described earlier, the reservation message is 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 is sent back to the receiver and the reservation message is discarded. Otherwise, if the reservation message requires a new reservation to be made, it propagates as far as the closest point along the way to the sender(s) where a reservation level equal to 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 reservation to a smaller, or equal, amount of resources than some previous request. As an example, assume H1 is a video source, and H4 has reserved enough bandwidth to receive the full video data stream while H5 wants to receive only low-resolution video data (Fig. 2). In this case, when the reservation request from H5 reaches S4, S4 makes the requested reservation over the link from S4 to H5 and then drops the request (i.e., does 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.

**EXAMPLE**

We consider a simple network configuration to illustrate in more detail how RSVP works. The network has five hosts connected by seven point-to-point links and three switches (Fig. 3). We assume that for links connecting hosts directly to a switch, the hosts act as switches in terms of reserving resources. To simplify the description, we assume adequate network resources exist for the path state or reserved resources. There is no path state or reserved resources for 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.

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.

**NO-FILTER RESERVATIONS**

Let us consider an audio conference among five participants, one at each of the five hosts (Fig. 3). In this case, each host behaves both as a source and a receiver at the same time. We make the following assumptions:

- The routing protocol has built a multicast routing tree so each sender can reach all the receivers.
- Each switch has received RSVP path messages (with the F-flag off from all the sources, so the switches do not record source information), and the complete path state for each switch has stored as described below, although in a real application sources may start at different times and the path state would be built up over time.
- No reservations have been made yet.

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 $R_1 (B, \text{no-filter})$ to S1, where $B$ is the amount of bandwidth needed to forward one audio stream. When S1 receives $R_1 (B, \text{no-filter})$, it first reserves resources over L1 (in the direction from S1 toward H1), then attaches the following reservation state to the path state to indicate the amount of the reservation made over L1.

<table>
<thead>
<tr>
<th>$S_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Incoming</strong></td>
</tr>
<tr>
<td>L1, L2, L6</td>
</tr>
<tr>
<td><strong>Outgoing</strong></td>
</tr>
<tr>
<td>L1, L2, L6</td>
</tr>
</tbody>
</table>

Finally, S1 forwards $R_1 (B, \text{no-filter})$ 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 $R_1 (B, \text{no-filter})$ sent along L6 reaches S2, which reserves B over L6 and forwards the message to links 5 and 7. When the copy of $R_1 (B, \text{no-filter})$ that was sent along L7 reaches S3, that switch reserves B over L7 and then forwards $R_1 (B, \text{no-filter})$ over links 3 and 4.

When H2 wants to create a reservation, it sends a reservation message $R_2 (B, \text{no-filter})$ to S1. Upon receipt of $R_2 (B, \text{no-filter})$, S1 first reserves B over L2, changing the path state to:

<table>
<thead>
<tr>
<th>$S_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Incoming</strong></td>
</tr>
<tr>
<td>L1, L2, L6</td>
</tr>
<tr>
<td><strong>Outgoing</strong></td>
</tr>
<tr>
<td>L1(B), L2(B), L6</td>
</tr>
</tbody>
</table>

S1 then forwards $R_2 (B, \text{no-filter})$ over L1 only, because it has forwarded 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 seven links in each of the two directions.

Before leaving this example of no-filter reservation, consider the tradeoff between keeping extra state information and the possibility of over-reserving resources on certain links. In the above example, we assumed all the path messages had the F-flag off, so no per-source information is kept at the switches. As a result, if each receiver requested 2B of bandwidth (i.e., an amount enough to carry two full audio streams), then 2B would be reserved on every link — even though on L1 (and similarly on L2, L3, L4, and L5) in the direction away from H1 we need only reserve 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. If the F-flag was 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 resources would have been reserved along the links.
Not maintaining per-source information can lead to an over-reserving of resources on some network links. However, in those applications involving many data sources with few resources required 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.

**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 messages 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 no reservations have yet been made. Thus, S1's path state contains the following entry:

<table>
<thead>
<tr>
<th>S1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outgoing-links: L2(src: H1, H4, S2, H5, S2) L6(src: H1, H5)</td>
</tr>
</tbody>
</table>

The notation L2(src: 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 a reservation message denoted R2 (B, H4), that is, H2 wants to receive packets only from source H4 and is reserving an amount B, sufficient for one source. The reservation message R2 (B, H4) reaches S1 via the L1 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.

S2's path state contains the following entries:

<table>
<thead>
<tr>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outgoing-links: L5(src: H1, H4, S2, H5, S2) L6(src: H1, H5) L7(src: H1, H5)</td>
</tr>
</tbody>
</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 toward H4.

S3's path state then has its entries changed to:

<table>
<thead>
<tr>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outgoing-links: L3(src: H1, S2, H4, H5, S2) L4(src: H1, S2, H5, S2) L7(src: H4, H4)</td>
</tr>
</tbody>
</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 some time afterward, H5 sends the reservation message R5(2B, *), where * indicates a request for dynamic-filter reservation. When S2 receives this reservation message, it reserves 2B over L5 (at least two 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 H4 terminates both receiving and sending without transmitting any teardown messages. As H4 no longer sends path or reservation-refreshes, all H4-related state will time out, changing the outgoing link entries in the various switches.

<table>
<thead>
<tr>
<th>S1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outgoing-links: L2(src: H1, H4, S2, H5, S2) L6(src: H1, H5)</td>
</tr>
<tr>
<td>S2</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>Outgoing-links: L5(src: H1, S1, H4, S3) L6(src: H5, H5) L7(src: H1, S1, H5, H5)</td>
</tr>
<tr>
<td>S3</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>Outgoing-links: L3(src: H1, S2, H5, S2) L4(src: H1, S2, H5, S2) L7(src: H4, H4)</td>
</tr>
</tbody>
</table>

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 the L7 direction.

For the sake of simplicity, in the above example we assumed each data stream requires the same bandwidth to forward. RSVP is designed to handle cases where each source may demand different amounts of resources, and each receiver may receive only a subset of the data from each source. In fixed-filter reservations, this requires each source filter be associated with a specific amount of resources. In dynamic-filter reservations, the receiver must receive the same amount of data when "switching channels."

**IMPLEMENTATION STATUS**

This article illustrates how RSVP works at a general level. For the sake of brevity and clarity, many details have not been presented; in particular, we have not described with any specificity the merging algorithm. We have, however, verified this design in a packet-level, interactive simulator, where all such details have been tested.

The simulator was written by one of the authors (LZ) and has been used in several previous simulation studies [8, 6, 14]. It 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 was implemented by Sugil Jamin (USC) for experimentation on DARNet, a cross-country T1 network testbed sponsored by...
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.

**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 [9, 15]. However, almost all of these prototypes deal exclusively with unicast reservations.

The Stream Protocol, ST [5], was a pioneering work in multicast reservation protocol design. ST was designed specifically to support voice and video 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 manage the tree establishment.

The successor to ST, ST-II [2], 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, so 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 elsewhere [16].

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 [17]. They share with us these viewpoints:

- 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.
- Furthermore, not to waste network resources, the filter(s) from all the receivers should be propagated toward the sender, so the subset of the data in which 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.

**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, several larger design issues remain unresolved, as detailed below.

RSVP was designed with minimal expectations of routing. Path states 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 remote lectures with several speakers at separate sites, one might want a dynamic filtered reservation where the filter is the same for each receiver, as proposed by Jacobson [18]. 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 an individual reservation state for each home.

**Summary**

RSVP's architecture is unique in that:

- It provides receiver-initiated reservations to accommodate heterogeneity among receivers as well as dynamic membership changes.
- It separates the filter from the reservation, thus allowing channel changing behavior.
- It supports a dynamic and robust multipoint-to-multipoint communication model by taking a soft-state approach in maintaining resource reservations.
- 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 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.

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

REFERENCES

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.