UNIVERSITY OF SOUTHERN CALIFORNIA

INFORMATION SCIENCES INSTITUTE
4676 Admiralty Way Marina del Rey, CA 90292

Tera-node Network Technology (TASK 1)

RSVPTNT - ReSerVation Protocol FINAL REPORT

Defense Advanced Research Projects Agency (DoD)
Computer Systems Technology Office
Under Contract #DABT63-95-C-0095
Issued by Directorate of Contracting
Fort Huachuca, AZ

REPORT DOCUMENTATION PAGE	*FORM APPROVED* *OMB NO. 0704-0188*

1. AGENCY USE ONLY	2. REPORT DATE March 14, 2000	3. REPORT TYPE AND DATES COVERED Final Report – 07/01/1995 – 06/30/1999

4. TITLE AND SUBTITLE Tera-node Network Technology (TASK1) ReSerVation Protocol (RSVPTNT)	5. FUNDING NUMBERS DABT63-95-C-0095
6. AUTHORS Robert Braden, Deborah Estrin, Steven Berson, Shai Herzog, Jeff Kann, Robert Lindell, Mohit Talwar, and Daniel Zappala	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) USC INFORMATION SCIENCES INSTITUTE 4676 ADMIRALTY WAY MARINA DEL REY, CA 90292-6695	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Directorate of Contracting ATTN: ATZS-DKO-I P.O. Box 12748 Fort Huachuca, AZ 85670-2748	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12A. DISTRIBUTION/AVAILABILITY STATEMENT UNCLASSIFIED/UNLIMITED	12B. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

RSVP, a setup protocol to create flow-specific reservation state in routers and hosts, is a component of the QoS extensions to the Internet architecture known as Integrated Services. RSVP was designed to provide robust, efficient, flexible, and extensible reservation service for multicast and unicast data flows. This report provides an overview of the development of RSVP, including an analysis of the key design decisions for the protocol as the protocol has evolved.

14. SUBJECT TERMS Internet Quality of Service, resource reservation, signaling protocol,soft state, multicasting.	15. NUMBER OF PAGES 18
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UNLIMITED

The Evolution of the RSVP Protocol

Robert Braden, Deborah Estrin, Steven Berson, Shai Herzog,
Jeff Kann, Robert Lindell, Mohit Talwar, and Daniel Zappala

March 14, 2000

USC/Information Sciences Institute

FINAL REPORT
Contract RSVPTNT-DABT63-95-C-0095
1 July 1995 - 30 June 1999

ABSTRACT: RSVP, a setup protocol to create flow-specific reservation state in routers and hosts, is a component of the QoS extensions to the Internet architecture known as Integrated Services. RSVP was designed to provide robust, efficient, flexible, and extensible reservation service for multicast and unicast data flows. This report provides an overview of the development of RSVP, including an analysis of the key design decisions for the protocol as the protocol has evolved.

1. Introduction

This document provides an overview of the development of RSVP (*ReSerVation Protocol*) Internet protocol. RSVP was designed as a setup or "signaling" protocol for Internet resource reservations. This document is a final technical report for the RSVP2 project at ISI under which ISI contributed to the design, standardization, and prototyping of RSVP.

The remainder of this section briefly recapitulates the historical context and basic requirements for RSVP. Section 2 provides an analysis of the key technical decisions as the protocol evolved since its beginning in 1991. Section 3 discusses two high-level design issues for RSVP, and Section 4 presents conclusions.

1.1 Historical Context

The development of Integrated Services and RSVP began in 1991. To place this event in context, in 1991 commercialization of the Internet was just beginning, the World Wide Web had not yet emerged, IP multicasting was still a research toy, and the OSI protocols still threatened to replace TCP/IP. However, the rapid growth of CPU power in silicon was already leading workstation manufacturers to forecast multimedia capabilities for future products. As a result, Internet researchers became concerned about the potential impact of multimedia traffic on the Internet [ISIP92]. The continuous flows of UDP-based multimedia traffic, which were not subject to congestion control, could prevent TCP connections from getting their fair share of bandwidth. Conversely, network congestion could prevent timely delivery of the real-time multimedia flows, providing poor service to multimedia users.

Two approaches to this problem were considered: (1) requiring that multimedia applications adapt to network congestion, and (2) explicitly supporting *quality of service* (QoS) for multimedia and other real-time traffic. The second requires that network resources be "reserved", to provide an upper bound on end-to-end delay and protect the real-time flows from each other and from best-effort traffic.

Adaptivity of multimedia applications to varying effective bandwidths is highly desirable in the statistically-multiplexed Internet. There were intense arguments about whether the evolution of fiber optics would lead to such an excess of bandwidth that congestion would never happen. However, the majority view was that there will always be significant parts of the Internet where low bandwidth or congestion will be outside the adaptive ranges for multimedia applications. There was furthermore an assumption that a multimedia user would rather receive a "busy" signal than have service degrade during a session. As a result, the decision was made to pursue both approaches: adaptive applications and Internet QoS.

ARPA-funded research beginning in the early 1970s had demonstrated packetized speech and developed a network protocol called ST-II [STII90] for real-time data delivery. In brief, ST-II implemented point-to-multipoint virtual circuits with QoS. However, there were three serious problems with ST-II as a general solution for Internet QoS. First, ST-II was incompatible with IP, so the adoption of ST-II would result in two parallel packet switching infrastructures. Second, ST-II used "hard" state, while the Internet community favors "soft" state, for robustness. Finally, ST-II built sender-oriented multicast trees, which presented serious scaling problems for a session with many multicast receivers.

1.2 Internet Integrated Services

The Integrated Services architecture was designed as a solution to supporting multimedia and other real-time traffic in the Internet [ISIP92, ISarch93]. The term "Integrated" refers to support for both real-time and best-effort traffic in the same packet-switched infrastructure.

Integrated Services is based upon three fundamental assumptions.

o It is still IP

The Integrated Services architecture is incremental, i.e., it is designed as a compatible extension of the original Internet architecture with its best-effort service.

Since reservations have a cost and many users will not want or need to pay that cost, best-effort service must always be available in the Internet. It was expected that this cost would be necessary only for those real-time applications that cannot adopt sufficiently well, and in those cases where there is not sufficient over-provisioning to provide adequate service.

o Multicast delivery is fundamental.

It was believed that teleconferencing and perhaps broadcast video might become "killer applications" for the Internet, and that efficiency would demand IP multicasting to support these applications.

o Signaling overhead is acceptable

The overhead for resource reservation setup, known as "signaling" in the world of telephony, should be small compared to the continuous multimedia packet streams

Integrated Services required the definition of new IP service models to provide QoS, and the implementation of these models meant resource reservations in routers and hosts. High-assurance isolation of real-time flows required fine-grained reservations, at the granularity of individual user flows, at least at the network edges. It was recognized that this could lead to an unacceptably large amount of reservation state in the "center" of the network, but it was believed that aggregate reservations would be acceptable in regions with a lot of reservation state [ISarch93].

Implementing Integrated Services required that new mechanisms be added to routers and end systems: admission control, packet classification, packet scheduling, and of course signaling to set up the reservations.

1.3 RSVP

The RSVP protocol was designed to be the signaling protocol to set up Integrated Services reservation state in hosts and routers [Zhang93]. The fundamental design of RSVP was developed by a research collaboration during the period 1991-1993. Beginning in 1993, a further research and development collaboration turned this proto-RSVP into a practical Internet protocol, and ISI constructed a prototype implementation of the evolving protocol. Since 1995, the RSVP protocol has been further refined and standardized by a Working Group of the IETF; the result is known as Version 1 of RSVP [RSVP97], which is now a Proposed Standard for the Internet.

The basic RSVP requirements followed from its role as the state setup protocol for Integrated Service

1. RSVP is an end-to-end protocol; requests come from applications and travel through routers.
2. RSVP is designed to make reservations for fine-grained flows
3. RSVP is designed from the beginning to support multicast as well as unicast flows. As shown in Figure 1 below, IP multicast establishes a multipoint-to-multipoint delivery tree for each session.

More generally, RSVP was designed with these objectives: logical simplicity, robustness, scaleability, flexibility, deployability, and extensibility. Section 2 describes how these objectives were met.

1.4 Integrated Services

RSVP is only one component of Integrated Services (although "RSVP" is often misused as a metaphor for Integrated Services). Other components that had to be developed included service models, link layer mappings, an API, and policy control.

o Service Models

Research on Integrated Services led to two alternative service models, Guaranteed service and Controlled Load service.

Guaranteed service [GuarA97] enforces a requested maximum end-to-end queueing delay (the speed of light is not under our control). It provides a high assurance service with a correspondingly high resource cost, which is most appropriate for real-time applications that are unable to adapt to network congestion.

Controlled Load service [CLoad97] provides a lower level of assurance -- a "soft" guarantee -- at a much lower resource cost than Guaranteed requires. Controlled Load essentially simulates an unloaded network, and it is suitable for adaptive applications, protecting them from extreme congestion.

Although there are many ways to implement these services, it may be helpful to think of Guaranteed service as using a weighted round-robin queueing discipline, while Controlled Load may be thought of as simple priority queuing.

o Link layer mappings

It was necessary to specify how to achieve the service models of Integrated Services over different link layer technologies. An IETF working group developed specifications for low-bandwidth point-to-point links (e.g., ISDN), for IEEE 802 networks, and for ATM circuits.

1.5 Differentiated Services

Since the Integrated Services research began in 1991, the problem has shifted. The World Wide Web has become the dominant Internet application. Multimedia flows are today only a relatively minor problem in the global Internet, although they may be significant within some corporate intranets. Today's major Internet problems are keeping up with explosive growth, supporting the World Wide Web, and providing the tools needed by commercial Internet service providers (ISPs).

In particular, it is of great commercial importance to an ISP to be able to sell different qualities of service to different customers. Integrated Services, which was designed to provide assured service (only) for continuous flows, is not a solution to this problem. It makes no sense to ask RSVP to set up a reservation for each transient TCP connection in a Web transaction, for example. The solution under development for this problem is Differentiated Services.

Differentiated Services allow an ISP to sell various grades of Internet Services. It explicitly avoids per-flow state in the network, except perhaps at the "edges". When possible, it makes resource allocations to large flow aggregates and over long time scales, replacing reservations with provisioning.

2. Overview of RSVP Design

The design of RSVP to meet the requirements of the previous section involved many technical decisions. If any of these decisions were changed, the protocol would change in a minor or a major way. We divide the decisions into those that seem most fundamental, in the following section, and those that are secondary, in Section 2.2. Section 2.3 summarizes RSVP's interfaces to other system components.

2.1 Primary RSVP Design Decisions

A. Receiver Initiation of Reservations (for scalability)

RSVP is receiver-oriented, so that it will scale to large multicast groups. As shown in Figure 1, RSVP reservation request *(Resv)* messages originate from receivers of data and travel upstream towards data senders. Note that the reservations are made at the upstream end of each link, although the reservation requests come from downstream.

It was suggested at several stages of RSVP design that it could support sender initiation as well as receiver initiation of reservations, since sender initiation appears simpler for the unicast case. However, the majority view was that any simplification that might result for unicast reservations was out-weighed by conceptual economy; there was a reluctance to define a protocol with two ways to do accomplish the same thing.

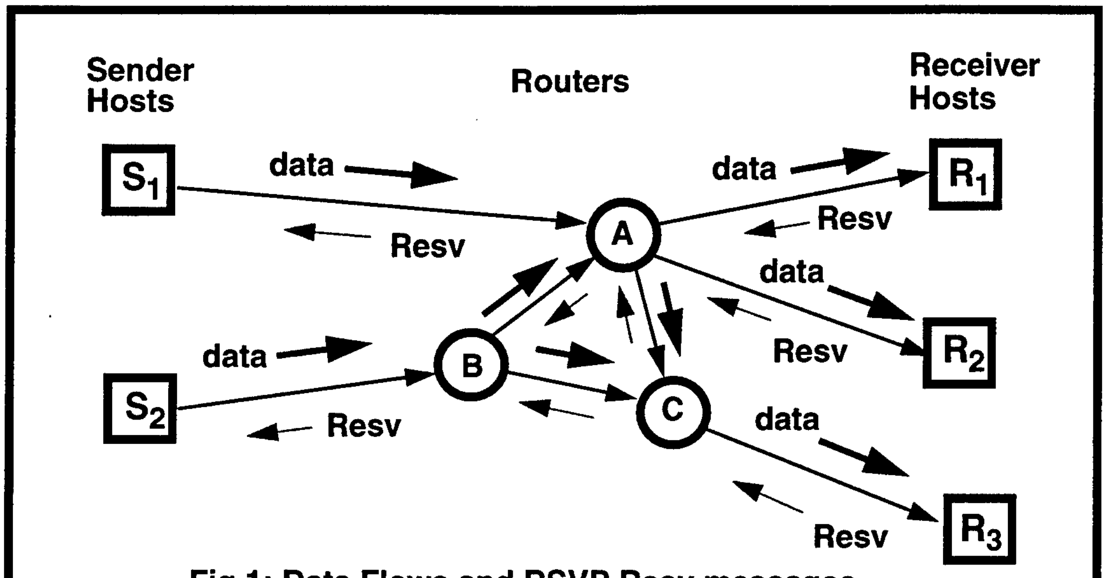

Fig 1: Data Flows and RSVP Resv messages for a multicast session.

B. Signaling distinct from routing (for deployability)

Resource reservations for a data flow must be made in each router along the path of the flow. The path is determined by some routing protocol whose distributed algorithm determines where to forward the data packets at each hop. It would be possible to develop a unified mechanism that encompassed both routing and reservation. However, this did not seem practical, since there already existed a variety of routing protocols in the Internet, and RSVP should be able to work with any of them.

The design decision was to adopt an incremental approach to combining reservation with routing [Zappala96]. Phase 0 is for RSVP to use existing routing protocols without change. In this initial phase, the routing protocol implementations would be modified only to allow the RSVP daemon to query about specific routes, to steer reservations along the route of the data flows.

The simple Phase 0 approach has two important problems. First, it may cause a reservation request to fail on the path provided by routing, when it might have succeeded along some alternate path. Second, if a route changes to a "better" route (but the original route still works), there may be excessive signaling overhead and perhaps brief signaling outages as reservations are reestablished over the new route. We say that the route over which a reservation has been made needs to be "pinned" [Zappala96].

To provide a partial solution to these problems, we can move to Phase 1. Here we use the current reservation-independent routing to choose a route, but allow RSVP to dynamically request an alternate path when the first one does not have enough resources [Zappala96]. Research at ISI showed that heuristics based on local information can be used to implement reasonably effective and scalable alternative path routing for multicast [Zappala00]. Such an alternative-path routing mechanism would be combined with route pinning in Phase 1.

The most general solution to the problem of unnecessary reservation failures would be "QoS routing", selecting a route that is known a priori to have sufficient resources available for the QoS requested for that packet flow. For example, one could imagine using all the resource measures into a big link-state calculation. This approach might work within an autonomous domain, but it is not generally scalable to the full Internet. A more feasible but less effective approach, known as "QoR" routing, would use a static metric like total bandwidth on each link, regardless of the current level of reservation. This provides routes that are likely but not guaranteed to have enough resources for requests.

At bottom, the problem of combining routing with resource reservations means building a common mechanism that encompasses both virtual circuits and connectionless packet forwarding. We further require the mechanism to be scalable to a very large network and to be able to find and pin the best path that will satisfy a reservation request. This general problem is very hard, and at present there is no totally satisfactory solution.

C. Soft state (for robustness and simplicity)

RSVP builds "soft" state, i.e., reservation state that times out unless it is periodically refreshed. A message that initially establishes a particular element of state or modifies an existing element is called a *trigger message*; subsequent messages are *refresh messages*. RSVP gains a great deal of simplicity by using identical messages for trigger and for refresh. When a route changes, the next refresh message uses the new path and becomes a trigger message for the new nodes. Thus, RSVP can automatically adapt to route changes with no additional mechanism. To change an established element of state, an end system can simply send a new trigger message with the modified state. This incremental state setup mechanism provides further simplicity.

The general algorithm for processing a statefull RSVP message is as follows:

```
if   (State element does not exist)
          { Create state element; Set timeout timer;}
else if (State in element and message don't match)
          { Update element to match message; Reset timeout timer;}
else  /* Refresh message */
          { Reset timeout timer; }
```

To handle merging of reservations (see item F. below), RSVP must generate refresh messages independently in each node. Hence, refresh messages do not typically propagate end-to-end, only hop-by-hop. However, a trigger message must generally be propagated immediately (but only as far as necessary; see discussion of merging, below), and it may travel as far as an end node.

Although soft state does provide automatic adaptation to route changes, it can result in a "liveness" problem. The default refresh period is relatively long, 30 seconds, and a node may increase this time still further to reduce refresh overhead on a congested link. This is a long time to wait for establishing a reservation on a new route when an old route has failed. To avoid this delay, RSVP contains a short cut mechanism called *local repair*. The routing protocols operating in the node signal to RSVP when a route changes, and RSVP then initiates refresh messages for all sessions using that route. There is a further complication that routing may not converge immediately, and therefore local repair is initiated after a short delay in some cases [RSVP97]. Although local repair should usually provide immediate reservations on the new path, the soft state refresh serves as a backup mechanism to ensure that the correct reservation state will be achieved eventually.

D. Transparent operation across non-RSVP routers (for deployability)

RSVP was designed to work transparently across an arbitrary "cloud" of non-RSVP routers. This is necessary to allow incremental RSVP deployment in the Internet (simultaneous global deployment being impossible). RSVP can be useful even when parts of an end-to-end path do not support Integrated Services, if those parts are sufficiently over-provisioned.

This apparently simple requirement of transparency had some profound consequences for the protocol. For example, it means that multiple Path messages (see B in Section 2.2 below) for a given session cannot be packed into a single message to decrease RSVP message overhead. Another potential problem is that the inability to protect RSVP message traffic from congestion through a non-Integrated Services cloud may cause delays in setting up RSVP state or intermittent loss of reservations. This setup delay problem is being mitigated by the addition of reliable RSVP message delivery [Berger00].

E. Support shared and distinct reservations (for flexibility)

A general RSVP session may have both multiple senders and multiple receivers, as shown in Figure 1. The rules that a router should use for mixing the input from different senders depends upon the application. For example, an audio stream is generally treated as a single shared channel, since the receiver cannot distinguish people talking at once. For each audio session, each node should contain a single reservation that is *shared* by all the audio senders that converge at that node.

On the other hand, there is no "silence" in video streams, and a receiver can display multiple pictures simultaneously on a computer screen. Each sender's video stream in a node therefore needs to be isolated from other streams. In other words, a separate or *distinct* reservation is needed for each video sender that merges at a node.

A basic RSVP requirement was to provide efficient support for both audio and video streams, so RSVP must support both *shared* and *distinct* reservations.

F. Heterogeneous reservations (for flexibility)

The decision was made that RSVP should support heterogeneous reservation requests, that is, different requests from different receivers. This decision led to considerable protocol complexity, and it is a design decision that may be changed in the future.

Suppose that the receivers R1 and R2 in Figure 1 send reservation request messages for different *flowspecs,* i.e., different QoS values, say Q1 and Q2. In practice, Integrated Services flowspecs are complex multi-dimensional vectors [RSVPuse97]. Router C must send a single reservation request upstream that is sufficient to encompass both requests, i.e., its flowspec must be the "max(Q1, Q2)". This max function is defined very precisely in the Integrated Services specifications [GuarS97, CLoad97]. The process of combining Q1 and Q2 in this way is called *merging* the reservation requests. Such merging is a fundamental aspect of RSVP.

One of the service models defined by Integrated Services allows a receiver to specify the maximum allowable queuing delay for the traffic [GuarS97]. Suppose that two receivers specify the same delay bound; this may result in different reservation flowspecs at some upstream merge point, due to different intervening paths. Therefore, RSVP must be able to merge heterogeneous requests to support the high-assurance Guaranteed service. In this service model, no packets are dropped at the merge point even though the "largest" flowspec is sent upstream.

2.2 Secondary RSVP Design Decisions

A. Soft-state refreshes for reliability

RSVP originally adopted the simplest mechanism for ensuring reliable delivery of its signaling messages: periodic refreshes for soft state supply any missing messages. Since there were no ACKs in the protocol, this mechanism provides only statistical reliability. This was considered sufficient because the protocol assumed that RSVP messages, like routing messages, would be given preferential QoS, so they would almost never be lost.

It has turned out in practice that this design decision was incorrect. It is not always possible to provide enhanced QoS for RSVP messages, due to non-RSVP clouds or to a shortage of CPU time on some routers. Furthermore, loss of an RSVP Resv message delays state establishment for one refresh period, which is likely to be of the order of 30 seconds or more. The result can be a serious failure of "liveness" for the protocol. Therefore, the RSVP Working Group of the IETF is currently standardizing a complex RSVP enhancement that includes reliable delivery of RSVP protocol messages, with explicit acknowledgments [Berger00].

Once reliable delivery is available, soft state refreshes are not logically necessary. It would be possible to modify RSVP to allow some nodes to use hard state, others soft state [JTW99]. This is a trade-off between refresh overhead and robustness. So far this suggestion has not been adopted in the IETF, and soft state still rules in RSVP.

B. Path messages

Some routing protocols (in particular, unicast routing protocols) do not calculate the previous hop for a route, only the next hop. Therefore, RSVP cannot generally query such a routing daemon for the previous data hop, in order to route a Resv message upstream towards the sender(s) of a data flow. To solve this problem, RSVP sends *Path messages* downstream, following the path of the data, for each flow. These Path messages are used to build a trail of *path state* for routing Resv messages upstream. Thus, in Figure 1 the arrows marked "data" could also be labeled "Path."

It was found that Path messages have several other uses besides routing Resv messages. Path messages carry information on the traffic profile as well as cumulative path information such as MTU, total propagation delay, and scheduling parameters [RSVPuse97]. Path messages also support the Guaranteed service model, to provide the effect of a two-pass reservation scheme [GuarS97].

C. Channel Switching

As we previously discussed, RSVP can support distinct reservations for each sender's flow, e.g, a video stream, all the way to the receivers. The user on the receiver may want to display all of the video images on a composite screen, or the user may select only a subset for display. However, it would wasteful to provide QoS for video streams across the network and then discard the streams at the receiver. RSVP was therefore designed to move the selection of sender streams to receive QoS as far as possible upstream towards the sender. To accomplish this, a Resv message specifies what senders should obtain the QoS indicated by the flowspec. In general, then, an elementary reservation request is a pair (Q, F), where Q is a flowspec specifying a QoS vector and F is a *filter spec* defining the packet subflow that is to benefit from the reservation. The filter spec is a parameter to the packet classifier, while the flowspec controls the corresponding packet scheduler queue. The receiver can change the F in the reservation request, to select a different sender. This may be likened to switching channels on a TV set.

In principle, RSVP filter specs could specify any combination of fields in any sequence of protocol headers. Early RSVP designs allowed a general mask-and-match definition for filters, but this was later simplified to a few specific filter formats. The basic filter spec format contains the source and destination IP address and the source and destination UDP/TCP port. Another format uses a IPv6 destination and source addresses and perhaps an IPv6 flow id. Finally, there are formats for using the SPI field of IP Security (IPSEC) headers in place of ports, when the ports themselves are obscured by end-to-end encryption.

All filter spec formats specify the IP address of the sender host, so we can think of the filter spec as selecting a particular sender. For audio flows, a shared reservation is used by all upstream flows; in this case, the elementary reservation has the form (Q, *), i.e., the sender selection is wild-carded.

These considerations show a few examples of the rich possibilities for reservation rules, especially for multicast flows. RSVP summarizes these rules in a parameter carried in Resv messages and called the reservation *style*.

The current version of RSVP includes three different reservation styles.

o *Wildcard Filter* (WF) style: reservation is shared by all upstream senders,

symbolized by: WF(Q, *)

o *Fixed Filter* (FF) style: distinct reservations for k particular senders,

symbolized by: FF((Q1, F1), (Q2, F2), ... (Qk, Fk))

o *Shared Explicit* (SE) style: one reservation shared among k particular senders,

symbolized by: SE(Q, F1, ... Fk)

Other more complex styles have been suggested [RSVP93].

D. Loop Prevention

The WF style seemed like a clever idea, providing a single shared reservation for all senders. It seemed to scale well; the Resv message size and the amount of classifier state for a WF message are independent of the number of senders [RSVP93].

Unfortunately, the research effort on RSVP revealed a serious flow in the WF style: it is subject to a subtle kind of self-refreshing loop. If the network topology contains a loop, certain combinations of senders and receivers can set up a circular refresh loop in which the reservation state persists even if when receivers stop requesting reservations [SCOPE96].

To avoid this problem, it was necessary to add a mechanism called a SCOPE list to a WF-style Resv message. The SCOPE list, which is in fact an explicit list of upstream senders, can be used to break the self-refresh loop [RSVP97, SCOPE96]. The SCOPE list causes the size of a WF-style Resv message to grow linearly with the number of senders, which voids one of the advantages of WF. However, a WF-style reservation still requires constant classifier state.

There was a discussion of dropping the WF style, avoiding the need for the SCOPE list, to simplify the RSVP protocol. The effect of a WF style reservation request can be achieved by an SE-style request that lists all senders (Path messages inform each receiver of the addresses of all senders). However, the WF style and its SCOPE list were retained because they result in constant-size classifier state.

E. Response to Admission Failures

Even though merging of heterogeneous requests is useful and perhaps necessary for some potential users of RSVP, it does raise a very serious and difficult fundamental problem: merging of reservations allows accidental or deliberate denial-of-service attacks on a receiver. For example, R2 in Figure 1 can request such a large reservation that the merged reservation forwarded by router A will certainly fail in router B. The result will be to deny reservations to R1 and R3 from sender S2. We call R2's action a *killer reservation.*

The RSVP protocol specification includes a complex mechanism to prevent killer reservations [RSVP97]. In brief, the effect of this mechanism would be to insert *blockade* state in router A that causes reservations from R2 to be omitted from the merge. Such blockade state periodically times out to adapt to a change in the requests.

A more general consideration of the killer reservation problem [Talwar99] reveals that there is a range of possible policies for making reservations when one or more nodes have inadequate resources for all the requests being merged. In the absence of merging, the policy is relatively simple: when admission control fails, the reservation is left in place downstream from the failure point but a Resv request is not propagated upstream from the failure point. With merging, the policies, and the mechanisms to enforce them, become much more complex.

Three policies that have been investigated are as follows [Talwar99].

o Global Best-Fit Policy

Under this policy, the single greatest reservation that can succeed in every node along the path(s) of the flow(s) is installed and maintained at every node. This policy is realized by the blockade state mechanism that is in the current RSVP specification.

o Local Best-Fit Policy

Under this policy, each node establishes the greatest of all downstream requests that can be accommodated, regardless of what reservation requests succeed at other nodes. This may result in reservations that vary from one node to the next along the path. An algorithm to realize this policy has been proposed [Talwar99].

o Plain Greedy Policy

This is similar to the Local Best Fit policy, except that if a node cannot accommodate the greatest downstream request, it installs the largest reservation it can provide.

The algorithms to realize these policies (including the blockade algorithm in use) typically rely on a transient period during which Resv messages travel upstream and the resulting ResvErr (admission control error) messages travel down, iterating until all nodes stabilize in the final state. The sequence of iterations can be long and complex [Talwar99].

F. Policy Control

Since resource reservations provide a subset of the user community with privileged service, there must be some technical, economic, or social mechanism to protect this privilege. That is, Integrated Services and RSVP require mechanisms for access control and accounting. We invented the neutral term "policy" for these two functions. A reservation request must generally pass both admission control, which ensures there are enough resources, and policy control, which ensures that the requestor is authorized to reserve these resources.

Unfortunately, policy control raises difficult problems of efficiency and functionality. The policy control mechanism must be general enough to implement a very wide range of possible policies, and it must be reasonably efficient and also secure. It may require bulky information such as cryptographic certificates.

The decision was made to carry policy control information "in-band" within the RSVP Resv and Path messages, rather than in a separate and parallel protocol. RSVP therefore carries policy information within Policy Data objects that are largely opaque to RSVP. However, the policy algorithms turned out to be entwined with the reservation algorithms of RSVP to a much larger extent than we expected [Policy00]. For example, there must be merging rules for policy data in Resv messages, although these rules are quite different from flowspec merging rules.

Since there was great uncertainty about the kinds of policies that would be needed, we decided to separate policy control from the rest of the RSVP reservation machinery. A short-term engineering decision was made that turned out to have profound consequences: we decided to "off-load" the policy control decision into a separate machine, linked to the router(s) running RSVP with a simple request/response protocol. It was expected that this would be a relatively short-term measure, and that policy control would eventually be folded back into the RSVP implementations. However, the policy control server quickly took on a life of its own in a separate IETF working group, and the simple request/response protocol became the COPS (Common Open Policy Service) [COPS00] protocol.

G. Extensibility

RSVP was designed to ease future extensions. Features to be added will require new protocol fields. For this reason, most logical elements of RSVP control messages are embedded in (type, length, value) encodings called "objects". The type is further subdivided into a generic type or "class" and a class-specific subtype or "C-type". This approach has provided great flexibility during protocol design and development.

H. Confirmations

If a reservation request fails, an RSVP error message is returned to the receiver(s) that initiated the request. However, the original protocol design had no positive confirmation message to indicate to the application that a reservation request was successful. Fully reliable confirmation would require global synchronization that is incompatible with RSVP's distributed soft state. There is also a problem defining confirmation semantics for the most general the multipoint-to-multipoint communication pattern that RSVP supports. However, a simple confirmation facility[RSVP97] was added to RSVP, upon request from application software vendors.

2.3 RSVP Interfaces

A central design problem for Integrated Services was how to modularize the components [ISarch93]. An important aspect of the design and specification of RSVP is to specify (but not over-specify) each of its major interfaces to other network software components. Figure 2 illustrates these interfaces within a node that is both an end system and a router.

o Interface to Routing

As indicated in Section 2.1, an RSVP daemon must have an interface to query the local unicast and multicast routing daemons for routes. ISI defined and prototyped a Phase 0 route query interface that is general enough to handle a range of multicast routing schemes [RSRR98].

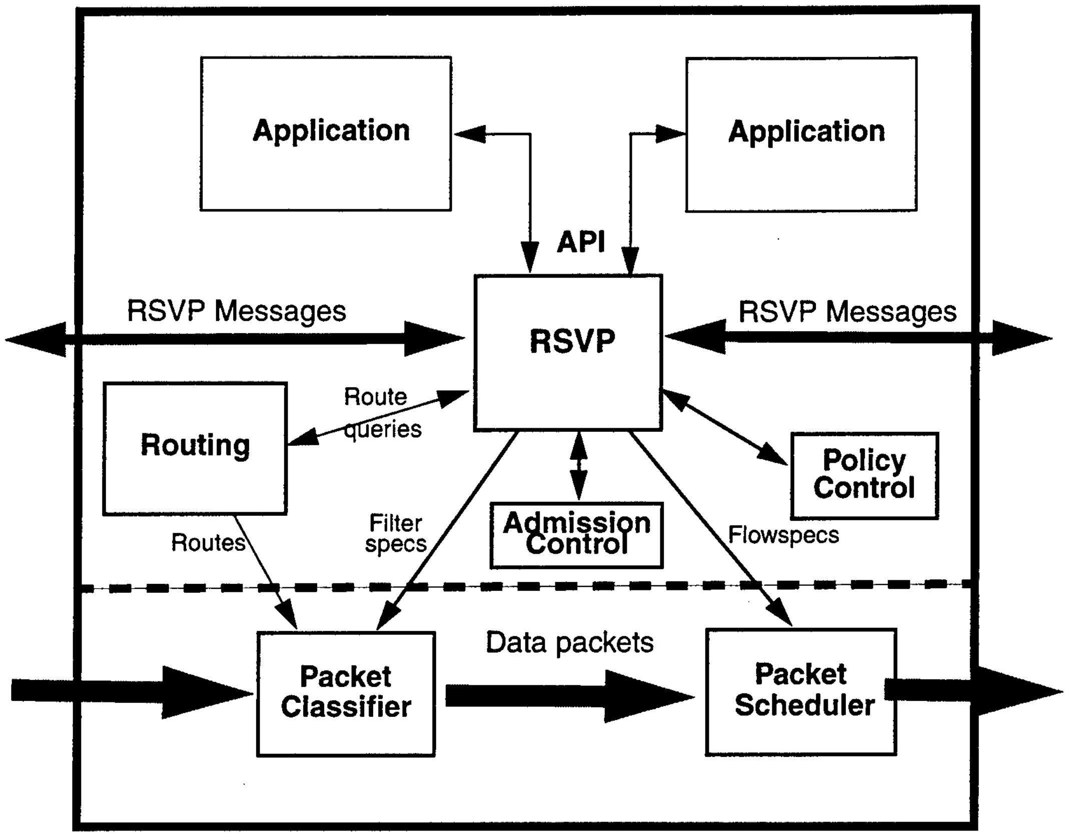

Figure 1: Illustration of RSVP Interfaces

o Interface to Traffic Control

"Traffic control" refers to the set of components in the packet forwarding path that implements the requested QoS. Traffic control includes admission control, packet classification, and packet scheduling. RSVP's job is to deliver parameters to traffic control at each node along the path of a data flow.

A generic RSVP interface to integrated services traffic control is defined in [RSVPspec96] and [RSVPuse96]. Details of RSVP's traffic control interface will be specific to the operating system and platform as well as the particular link layer technology in use.

The QoS parameters in a Resv message are bundled into a flowspec object, whose complex internal encoding and semantics [RSVPuse97] are largely opaque to RSVP. RSVP needs only very narrowly specified operations on flowspecs, most importantly the merging algorithm. Adding a new service model should require only minimal and clearly-defined modifications to an RSVP implementation.

o Interface to Link Layer protocol

The "natural" protocol model for RSVP is based upon simple point-to-point links. However, RSVP must also handle both broadcast media and switched virtual-circuit media (most particularly, ATM) at the link layer. The ATM case was the most complex [ATM98]. Requesting an RSVP reservation across an ATM cloud must result in opening a virtual circuit at the ATM layer, with an ATM QoS mapped appropriately from the RSVP flyspeck.

Note that running RSVP over IP over ATM creates a clash of fundamental models. RSVP is already trying to impose (loose) connection semantics onto a datagram (connectionless) network service; running IP over ATM means carrying that datagram service over a connection-oriented infrastructure. Hence, the RSVP/ATM design problem effectively involves connections over datagrams over connections, with QoS at two different layers and following different service models. As always, multicast makes the entire problem much harder [ATM98].

Handling this variety of link layer models resulted in modularity issues in our prototype implementation, which were never satisfactorily solved.

o **Interface to Policy Server**

As noted earlier, RSVP must implement the COPS protocol for communicating with a policy server [COPS00].

o **Interface to Applications: API**

Finally, applications need a standard interface to request resource reservations. ISI and Sun Microsystems originally developed the RAPI (RSVP API) interface [RAPI98], which is also contained in generic form in the RSVP protocol specification [RSVP97].RAPI has since been standardized by Open Group. RAPI allows an application to specify reservation requests in parallel with the API for sending data, e.g., a socket interface. The application is expected to be able to coordinate the two interfaces.

Later, Microsoft developed an alternative API for RSVP in Winsock2; this API incorporates reservation requests into the data socket calls.

3. Major RSVP Design Issues

We now turn to a discussion of two global design issues for RSVP: overhead and generality.

3.1 RSVP Overhead

The major components in RSVP overhead are the per-packet and per-byte cost of transmitting RSVP messages, the CPU time for processing RSVP messages, and the memory required for RSVP state.

Minimal RSVP Resv messages are of the order of 150 bytes in length for IPv4, or 220 bytes for IPv6 (including IP header, an MD5 Integrity object and one sender). Specifying more senders increases the size. For IPv4, a 1500 byte Resv message can specify 340 senders for a WF style reservation or 28 senders for an FF style reservation.

In principle, individual RSVP messages can grow very large if policy data objects are very large (e.g., because large cryptographic certificates are required) or if there is a very large list of senders. The current RSVP specification [RSVP97] supports only IP fragmentation, limiting an RSVP message to 64KB. Unless policy data objects are very large, this limit should support every reasonable scenario of RSVP usage. For example, for IPv4 a WF style reservation could specify up to 16000 senders, or an FF style reservation could specify up to 1360 senders.

o **RSVP Message Cost**

Transmission of RSVP control messages requires some link bandwidth.Sending a minimal RESV refresh message each 30 seconds uses approximately 40 bits per second (60 bps for IPv6) per link per session. If we use the current engineering design point of 10,000

RSVP sessions, there will be 333 refresh messages or 400K bits per second per link. A minimal Path message will be slightly smaller but will generate the same order of magnitude of traffic.

In some circumstance, at least, this volume of messages may be a significant load. These numbers help motivate a package of RSVP optimizations that is currently being standardized [Berger00]. This package includes the ability to send a short message identifier in lieu of an entire Path or Resv refresh message, reducing the refresh message traffic by a factor of roughly 30. Since they were added after the bulk of RSVP protocol design and since they have complex interactions with many existing RSVP features, these optimizations have some complex and undesirable interaction with other RSVP design features. For example, freedom from packet loss is gained at the loss of automatic accommodation to route changes.

o **RSVP Processing Cost**

The bottleneck resource is probably the router CPU for processing the messages, rather than the network bandwidth for transmitting them. The optimization package [Berger00] should significantly reduce message processing cost.

o **RSVP Memory Cost**

RSVP requires additional memory in hosts and routers for: (1) RSVP path and reservation state, (2) the resulting traffic control state in classifier, packet scheduler, and admission control.

RSVP path and reservation state increases linearly with the number of sessions. Based upon ISI's prototype implementation of RSVP, a rough estimate of RSVP state memory is 600 bytes per unicast session. A corresponding estimate for a multicast session is 120 + 180*S + 260*Ni bytes per session, where S is the number of senders and Ni is the number of outgoing interfaces on which reservations have been made.

No such simple rule can be given for traffic control state, although it should increase no worst than linearly, and perhaps only logarithmically, with the number of sessions.

During the original Integrated Services research effort, state storage was not expected to become a major scaling problem, because the expected application was multimedia that would include video. In that case, the required aggregate bandwidth would limit the number of sessions and therefore the amount of state. For example, suppose there are one thousand multimedia conferences, each using 1 Mbps for a video flow; this creates a gigabit of data traffic but requires of the order of only 1 MB of RSVP state storage. However, recently the primary application of RSVP and Integrated Services has been shifting to Internet telephony. Suppose each call requires 30Kbps; then it takes roughly 30,000 sessions to fill the same gigabit pipe, and this implies of the order of 30MB of RSVP state space. Experts differ on whether 30MB of memory on a large router is excessive.

In summary, the processing and bandwidth required for RSVP will be brought into feasible ranges by a package of protocol optimizations that is nearing standardization. The solution to the state space problem is discussed in the next section.

3.2 QoS Flow Aggregation

A familiar indictment of RSVP has been that "it doesn't scale." In a sense, it scales perfectly, in fact: the overhead is strictly proportional to the number of flows. However, the linear factors matter.

This scaling problem was recognized during the early Integrated Services research, which planned to solve it by aggregating reserved flows in the middle of the network [ISarch93]. In regions where there are many Integrated Services flows, statistical aggregation can be relied upon to provide the desired end-to-end service. In these regions, individual flows can be mapped into a small fixed number of service classes. Differentiated Services provides a natural tool for this purpose.

The general plan that is emerging for full Internet-scale deployment of Integrated Services therefore has the following components:

o RSVP will be used end-to-end, making fine-grained reservations for Integrated Services at the edges of the network.

o Differentiated Services will be used in the middle of the network.

o The border routers to the Differentiated Services region will maintain full RSVP state. The ingress routers will aggregate the reserved flows into the appropriate Differentiated Services classes, and the egress routers will dis-aggregate the traffic back into individual RSVP flows.

o RSVP messages will pass transparently through the center of the network.

Multicast makes some of the details of this approach are much harder. For multicast, it seems necessary to maintain limited partial state within the center of the network, to handle heterogeneity [Aggr98].

3.3 A General Internet Signaling Protocol

The design of RSVP attempted to ease future extensions. An RSVP message has the general form:

RSVP message = {header} {typed "object"}*

As explained earlier, each RSVP "object" is a (type, length, value) encoding of some protocol data item. This approach has provided great flexibility during protocol design and development. During RSVP development, many changes and enhancements involved the definition of new C-types and/ or new generic types.

Besides this syntactic issue, protocol extensions raise another problem: RSVP extensions should be backwards compatible with existing implementations. That is, when a RSVP implementation receives a object (or even a message type) that it does not recognize, it should still take the correct default action. This turned out to be a hard problem, and the present RSVP design [RSVP97] has only a partial solution.

Many of the design details of RSVP are quite specific to its original planned usage, making resource reservations for Integrated Services. However, it is possible to look at RSVP in less detail and see a generic soft-state signaling protocol for the Internet. Several other RSVP working groups have adopted or adapted RSVP for their own signaling needs.

o The RSVP general design has been adapted to do layer-2 signaling in the Subnet Bandwidth Manager, for mapping Integrated Services into IEEE 802 networks.

o RSVP is being extended for use in setting up label-switch paths for Multiprotocol Label Switching (MPLS), which is effectively layer 2.5.

o There is discussion of using RSVP for configuring virtual private networks (VPNs).

o There has been discussion of using RSVP for setting up explicit routes.

RSVP is appealing for these new applications because of the syntactic ease of defining new object classes, and because the general soft-state model is widely applicable. However, the number of RSVP object types is rapidly increasing, and it is very unclear how to define interoperable subsets of the many functions that are being assumed by RSVP.

A worthwhile future research topic would be to explore dividing the current RSVP into two layers: a generic soft-state setup protocol and a layer that contains the semantics for a specific signaling application. Then the original RSVP would consist of a generic layer and a layer that is specific to Internet Integrated Services.

4. Conclusions

We have shown that RSVP met its general goals -- logical simplicity, robustness, scaleability, flexibility, adoptability, and extensibility -- to a large degree, but not entirely. Although the basic soft-state model provides considerable simplicity, the RSVP design accreted many "features" before it was complete. Such features include SCOPE lists, blockade state, (necessarily) both IPv4 and IPv6 support, IPSEC support, reliable delivery, refresh overhead reduction, and handling some arcane cases of IP multicast.

ISI personnel supported by this project edited the primary RSVP specification document [RSVP97], co-chaired the RSVP Working Group [AS97], and collaborated with other groups to make research contributions on RSVP's routing interface [RSRR98, Zappala96, Zappala00], the ATM interface [ATM98], policy control [Policy00], WF-style looping [Scope96], the killer reservation problem [Talwar99], the API [RAPI98], and aggregation [Aggr98]. ISI also produced a widely-used reference implementation of RSVP.

5. References

[Aggr98] Berson, S. and S. Vincent, *Aggregation of Internet Integrated Services State*. IETF Work in Progress, August 1998. Available from http://www.isi.edu/rsvp/pub.html. An earlier version of this paper was presented at IWQoS '98.

[AS97] Mankin, A., Baker, F., Braden, B., Bradner, S., O'Dell, M., Romanow, A., Weinrib, A., and L. Zhang, *Resource ReSerVation Protocol (RSVP) -- Version 1 Applicability Statement.* RFC 2208, September 1977.

[ATM98] Crowley, E., Berger, L., Baker, F., Borden, M., and J. Krawczyk, *A Framework for Integrated Services and RSVP over ATM.* RFC 2382, August 1988.

[Berger00] Berger, L., Gan, D., Swallow, G., Pan, P., and Franco Tommasi, *RSVP Refresh Overhead Reduction Extensions.* IETF Work in Progress, January 2000.

[CLoad97] Wroclawski, J., *Specification of the Controlled-Load Network Element Service.* RFC 2211, September 1997.

[COPS00] Boyle, J., Cohen, R., Durham, D., Herzog, S., Rajan, R., and S.Sastry, *The COPS (Common Open Policy Service) Protocol.* RFC 2748, January 2000.

[Crypto00] Baker, F., Lindell, R., and M. Talwar, *RSVP Cryptographic Authentication.* RFC 2747, January 2000.

[Design96] Braden, R., Estrin, D., Berson, S., Herzog, S., and D. Zappala, *The Design of the RSVP Protocol.* Final Technical Report for DARPA Contract DABT63-91-C-0001, June 1996.

[Diag00] Terzis, A., Braden, B., Vincent, S., and L. Zhang, *RSVP Diagnostics Messages.* RFC 2745, January 2000.

[GuarS97] Shenker, S., Partridge, C., and R. Guerin, *Specification of Guaranteed Quality of Service.* RFC 2212, September 1997.

[IPSEC97] Berger, L. and T. O'Malley, *RSVP Extensions for IPSEC Data Flows*. RFC 2207, September 1977.

[ISIP92] Clark, D., Shenker, S., and L. Zhang, *Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanisms*. Proc. SIGCOMM '92, Baltimore, MD, August 1992.

[ISarch93] Braden, R., Clark, D., and S. Shenker, *Integrated Services in the Internet Architecture, an Overview*. RFC1633, October 1993.

[Policy00] Boyle, J., Cohen, R., Durham, D., Herzog, S., Rajan, R., and S.Sastry, *COPS Usage for RSVP.* RFC 2749, January 2000.

[RAPI98] Braden, R. and D. Hoffman, *RAPI -- An RSVP Application Programming Interface*. IETF Work in Progress, August 1998. Available from: http://www.isi.edu/rsvp/pubs.html.

[RSRR98] Zappala, D. and J. Kann, *A Routing Interface for RSVP.* IETF Work in Progress, June 1998. Available from: http://www.isi.edu/rsvp/pubs.html.

[RSVP97] Braden, R. (Ed.), Zhang, L., Berson, S., Herzog, S., and S. Jamin, *Resource ReSerVation Protocol -- Version 1 Functional Specification*. RFC 2205, September 1997.

[RSVPuse97] Wroclawski, J., *The Use of RSVP with IETF Integrated Services*. RFC 2210, September 1997.

[Scope96] Zappala, D., *RSVP LOOP Prevention for Wildcard Reservations*. ISI internal document, February 1996. Available from http://www.isi.edu/rsvp/pub.html.

[ST-II90] Topolcic, C., *Experimental Internet Stream Protocol -- Version 2 (ST-II).* RFC 1190, October 1990.

[Talwar99] Talwar, M., *RSVP Killer Reservations*. IETF Work in Progress, January 1999. Available from http://www.isi.edu/rsvp/pub.html.

[Wroclawski99] Wroclawski, J., Private Communication. March 1999.

[Zappala96] Zappala, D., Braden, B., Estrin, D., and S. Shenker, *Interdomain Multicast Routing Support for Integrated Services Networks*, White Paper, December 1996. IETF Work in Progress, March 1997. Available from http://www.isi.edu/rsvp/pub.html.

[Zappala00] Zappala, D., *Alternate Path Routing for Multicast*. To appear in IEEE Infocomm 2000, March 2000.

[Zhang93] Zhang, L., Deering, S., Estrin, D., Shenker, S., and D. Zappala, *RSVP: A New Resource ReSerVation Protocol.*, IEEE Network, September 1993.