Reservations for Aggregate Traffic: Experiences from an RSVP Tunnels Implementation

Andreas Terzis, Lixia Zhang
University of California, Los Angeles, CA
Ellen L. Hahne
Bell Laboratories, Holmdel, NJ

Abstract

Among its various uses, IP-in-IP tunneling is a simple way to aggregate the data flows from multiple sources to multiple destinations into one flow, to cross part of the Internet. In this paper we report our design and implementation of RSVP support for resource reservations over IP-in-IP tunnels, and our experience from this effort that revealed a number of issues related to making resource reservations for aggregate data flows. First, aggregation and de-aggregation go in pairs, thus the exit point of the tunnel must have adequate information to be able to de-multiplex the aggregate tunnel reservation back to reservations for individual flows. Second, if multiple reserved sessions exist over one tunnel, the two tunnel end points need mechanisms to synchronize on which end-to-end reservation is bound to which tunnel reservation; on the other hand mapping all reservations of the same traffic class into one tunnel session can substantially simplify the protocol. Furthermore, one must also properly map error reports from the aggregate reservation back to the ends of individual flows.

1 RSVP Tunnels

1.1 Encapsulation Tunnels

Many large corporations are moving their networking infrastructure away from leased lines to the public Internet, as a way of cutting down communication costs. Responding to this market demand, most nation-wide Internet Service Providers offer today Virtual Private Network (VPN) services.

A VPN provides the illusion of a private network overlaid over the public Internet. In a VPN scenario, the company's sites are connected over the public Internet with virtual links called "tunnels". Packets that have to cross sites enter one end of the virtual link and are transported, through the common infrastructure, to the other end of the link.

IP tunneling uses a simple encapsulation technique. The tunnel "entry" point adds an outer IP header in front of the original IP header of each packet, givning all intermediate routers an illusion that packets are delivered between the outer source and destination addresses. When the encapsulated packet reaches the tunnel exit point, the outer header is thrown away and the packet is further routed towards its final destination.

VPNs seem a prime environment for the deployment of QoS support (such as Integrated Services and RSVP [3]) since they support business critical applications and are replacing a more controllable environment (the private network created by leased lines).

Using the current IP-in-IP encapsulation model, RSVP reservations are not possible. Since all the packets that reach one of the tunnel end-points are encapsulated before being sent to the other side, two main problems arise. First, the end-to-end RSVP messages become invisible to intermediate RSVP-capable routers residing between the tunnel end-points. Second, the usual RSVP filters cannot be used, since data packets are also encapsulated with an outer IP header, making the original IP (and UDP or TCP) header(s) invisible to intermediate routes between the two tunnel end points.

1.2 Mechanism Description

The proposal of providing RSVP support over IP-in-IP Tunnels [5] was born from the need to support resource reservations over IP-in-IP tunnels.

Fig. 1 shows a simple tunnel topology, where the senders and receivers of an RSVP session are connected through a tunnel between R_{entry} and R_{exit}. An RSVP session may be in place between R_{entry} and R_{exit} to provide resource reservation over the tunnel. We refer to the first RSVP session as the *end-to-end* session, and the second the *tunnel* RSVP session.

A tunnel RSVP session may exist independently from any end-to-end sessions. One may create, for example through some network management interface, an RSVP session over the tunnel to provide QoS support for data flows from S_1 to R_1, even when no end-to-end RSVP session exists between S_1 and R_1.

When an end-to-end RSVP session crosses an RSVP-capable tunnel, there are two possibilities to support the end-to-end reservation over the tunnel: mapping the end-to-end session to an existing tunnel RSVP session, and creating a new tunnel RSVP session. In either case, the picture looks like a recursive application of RSVP. The tunnel RSVP ses-

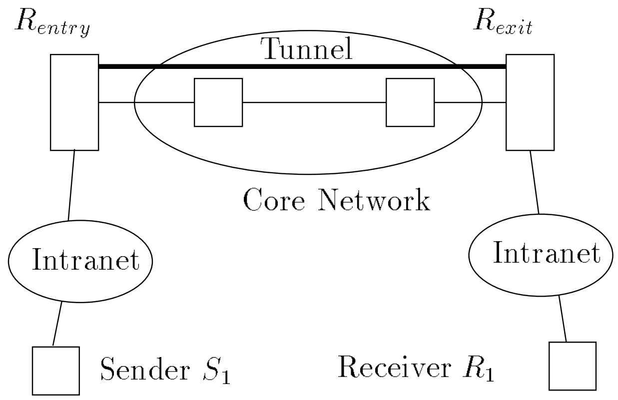

Figure 1: RSVP-Tunnels Model

sion views the two tunnel endpoints as two end hosts with a unicast Fixed-Filter style reservation in between. The original, end-to-end RSVP session views the tunnel as a single (logical) link along the path between the source(s) and destination(s). The PATH and RESV messages of the end-to-end session are encapsulated at one tunnel end-point and get decapsulated at the other end, where they get forwarded as usual.

In both cases, it is necessary to coordinate the actions of the two RSVP sessions when both exist, to determine whether or when the tunnel RSVP session should be created and torn down, and how to correctly map the errors, Adspec and other reservation related information from the tunnel RSVP session to the end-to-end RSVP session. The association between the end-to-end and the tunnel sessions is conveyed through the newly defined SESSION_ASSOC object in the tunnel PATH messages. This object associates an end-to-end session to a tunnel session. The tunnel exit point R_{exit} records this association, so that when it receives reservations for the end-to-end session, it translates them to reservations for the corresponding tunnel session.

Treating the two tunnel end-points as a source and destination host, one can easily set up a FF-style reservation between them. Now the question is what kind of filterspec to use for the tunnel reservation, which directly relates to how packets get encapsulated over the tunnel. We discuss two cases below.

If all the packets traversing a tunnel can use the reserved resources, then the current IP-in-IP encapsulation could suffice. The RSVP session over the tunnel simply specifies a FF style reservation with R_{entry} as the source address and R_{exit} as the destination address and zero as source and destination ports.

However if only part of the packets traversing the tunnel can use the reservation, we encapsulate the qualified packets not only with an IP header but also with a UDP header. This allows intermediate routers to use standard RSVP filterspec handling without knowing the existence of tunnels.

To simplify the implementation by reducing special case checking and handling, we decided that all data packets using reservations are encapsulated in IP+UDP. The source port for the UDP header is chosen by the tunnel entry point R_{entry} when it establishes the initial PATH state for the new tunnel session. The destination UDP port used in tunnel sessions is a well known port, assigned by IANA.

1.3 Session Association and Error Mapping

In the previous section we described two possibilities to to associate end-to-end sessions with corresponding tunnel sessions: mapping all the end-to-end sessions of the same traffic class to a single tunnel session, or allowing multiple sessions for each traffic class, with an extreme case of creating a one-to-one mapping between end-to-end sessions and tunnel sessions. In general, deciding which end-to-end sessions map to which tunnel sessions is a policy issue that is up to the network managers to decide. Numerous other modes of aggregation are also possible, for example aggregating all traffic for one customer. However we notice that, if we limit RSVP to support only the simplest mapping of all the end-to-end sessions of the same traffic class to a single tunnel session, we eliminate the need for the SESSION_ASSOC object which is exchanged between the two tunnel ends to associate end-to-end sessions with corresponding tunnel sessions.

Another type of mapping between the two levels of RSVP sessions is error reports. When R_{exit} receives a RESV for an end-to-end session, it first sends or refreshes (with possibly changed parameters) the corresponding tunnel RESV message and waits for a confirmation from R_{entry} that the reservation was successful before forwarding the end-to-end reservation request. If R_{exit} immediately forwarded the end-to-end request over the tunnel, then if the tunnel reservation failed, it would have to explicitly tear down, the installed reservation "past" R_{entry}. When a tunnel session RESV request fails, an error message is returned to R_{exit}. R_{exit} must treat this as an error crossing the logical link and forward the error back to the receiver.

2 Scaling Issues

Recently, a concern has been raised regarding RSVP's scalability. Since RSVP reservations are initiated by individual applications, which are identified by their IP addresses, protocol, and port numbers, all the RSVP routers on the path from the senders to the receivers have to keep state per application data flow. This can be burdensome, especially for backbone routers that connect to high speed links and may carry hundreds of thousands of flows simultaneously.

Some recent works [4], [1], [2] have identified this problem and suggested some possible solutions. In the rest of this section, we discuss the scaling properties of RSVP reservation over tunnels. We show that tunnel reservations, when used properly, can substantially reduce the amount of RSVP control state at backbone routers and reduce the number of RSVP messages exchanged across backbone routers.

2.1 State reduction

When one makes RSVP reservation over tunnels, state aggregation is achieved for the intermediate routers because they are no longer aware of end-to-end RSVP sessions, instead they handle only the RSVP messages generated by R_{entry} and R_{exit}.

The larger the degree of aggregation at the tunnel endpoints the larger the gain in reduced RSVP state in the network backbone routers. At one end of the spectrum, we have individual end-to-end RSVP sessions getting mapped to separate tunnel senders, thus achieving no state reduction in the intermediate routers. At the opposite end of the spectrum, we can achieve the largest amount of state aggregation possible by mapping all end-to-end sessions of the same traffic class to a single tunnel session. In this case intermediate routers in the tunnel only see one aggregate session per tunnel per traffic class.

2.2 Reducing the number of messages

Along with the memory requirements, the other source of overhead imposed on routers is the processing of RSVP messages.

Let us consider the RSVP message exchanges for the tunnel sessions. Although end-to-end RSVP messages are still sent through the tunnel, due to encapsulation they are invisible to intermediate routers in the tunnel and therefore require no RSVP processing.

In the case of tunnel PATH messages, R_{entry} collects the PATH information from all the senders of end-to-end sessions that map to the same tunnel session and sends one PATH message per tunnel session per refresh period. The Tspec in the tunnel PATH message is equivalent to the sum of Tspecs of all the senders belonging to end-to-end sessions mapped to the specific tunnel session.

According to [5], end-to-end RESV messages will trigger an immedidate tunnel RESV message only if they represent changes from the originally reserved value, which presumbly do not happen very often. We can further reduce the tunnel RESV message frequencys by changing the above rule so that the tunnel end points can only send RESV messages for specific increments (for example only in the order of hundreds of kilobits), then R_{exit} will send additional tunnel RESV message only when the aggregate amount from end-to-end reservations changed by more than this threshold value. We call this scheme, the *threshold* scheme.

The threshold scheme does not affect the soft-state character of the protocol. After a crash R_{exit} will have to reacquire the PATH state and send RESV messages to restore the amount of reserved resources in any case. Once R_{entry} becomes alive after the crash, the end-to-end RESV messages will drive the amount of the tunnel reservations to the level that existed before the crash.

3 Conclusions

As large corporations move their networking infrastructure away from leased lines to the public Internet, IP-in-IP tunnels are becoming a common tool used by the Internet Service Providers (ISPs) to build Virtual Private Networks (VPNs) to meet such demands. We have designed and implemented RSVP support for resource reservations over IP-in-IP tunnels, which can be used to provide VPNs with quality of service guarantees.

This undertaking not only produced a useful tool, but also helped us identify several issues related to making resource reservations for aggregate data flows. First, aggregation and de-aggregation go in pairs, thus the exit point of the tunnel must have adequate information to be able to de-multiplex the aggregate tunnel reservation back to reservations for individual flows; one way to convey the information about individual end-to-end reservation information to the exit point of the tunnel is to encaptulate end-to-end RSVP messages through the tunnel. Second, if multiple reserved sessions exist over the tunnel, the two tunnel end points need mechanisms to synchronize on which end-to-end session is bound to which reservation over the tunnel; on the other hand mapping all flows of the same class into one tunnel session can substantially simplify the protocol. Third, one must also properly map error reports from the tunnel back to the end user reservations.

We believe these are common issues one encounters when making reservations for aggregate flows, and our approaches in RSVP tunneling implementation can be applied to more general cases.

4 Acknowledgments

This work is partially supported by a research grant from Intel Corporation. Part of this work was done while Andreas Terzis was a summer intern at Bell Labs last year. The authors would like to thank Henning Schulzrinne for numerous discussions of RSVP tunnels as well as the anonymous reviewers for their insightful comments.

References

[1] S. Berson and S. Vincent. Aggregation of Internet Integrated Services State. *Internet-Draft, work in progress*, November 1997.

[2] J. Boyle. RSVP extensions for CIDR aggregated data flows. *Internet-Draft, work in progress*, June 1997.

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

[4] R. Guerin, S. Blake, and S. Herzog. Aggregating RSVP-based QoS Requests. *Internet-Draft, work in progress*, November 1997.

[5] L. Zhang, J. Wroclawski, J. Krawczyk, and A. Terzis. RSVP Operation Over IP Tunnels. *Internet-Draft, work in progress*, February 1998.