RSVP and Integrated Services in the Internet

Dilip Kandlur * Ashish Mehra † Debanjan Saha ‡

Abstract

This paper presents the design and implementation of a quality of service architecture for the Internet. The architecture is based on the emerging standards for resource reservation in the Internet, namely the RSVP protocol and the associated service specifications defined by the Internet Engineering Task Force (IETF). Our architecture represents a major functional enhancement to the traditional sockets based communication subsystem, while preserving application programming interface and binary compatibility with existing applications. It is scalable and supports a wide variety of network interfaces ranging from legacy LAN interfaces, such as Token Ring and Ethernet, to high-speed ATM interfaces.

1 Introduction

As audio and video annotations become common features on Web pages and as applications like InternetPhone, NetRadio, and WebTV become ubiquitous on the Internet, the need for "better than best effort" network connectivity will become inevitable. To address this, the Internet Engineering Task Force (IETF) is developing a set of protocols and standards for Integrated Services on the Internet [3, 6, 2, 5]. In the IETF's vision, and one that we share, applications can request and reserve resources in the network and at the hosts using an end-to-end Resource ReSerVation Protocol (RSVP) [7, 4]. Resource management is performed via per-flow traffic shaping and scheduling for various classes of service [3].

In order to support integrated services on the Internet, the network routers as well as end hosts need to be enhanced to perform classification of traffic on a per-flow basis, create and maintain flow specific reservation soft states, and handle data packets from different flows in accordance with their service requirements. In this paper we focus on resource management, protocol stack extensions, and device support required at the end hosts to enable RSVP-based quality of service (QoS) infrastructure in the Internet. More specifically, we concentrate on the design and implementation of QoS support on Unix-like[1] Internet servers, which are the typical source of multimedia data on the Internet.

The heart of our quality of service architecture is a new kernel module called QoS Manager. It is entrusted with managing communication related system resources at the end-hosts. Applications or their designated agents can request reservations on a network session to the QoS Manager. Typically, applications would use the RSVP application interface for reserving network resources along the path of the data flow and the RSVP agent would act as the designated agent to request local resource reservations from the QoS Manager. We have extended the socket API and created a new protocol family for applications to avail the services of the QoS Manager. In response to a reservation request, the QoS Manager, in cooperation with the network device driver, memory allocator, and the network interface handlers (IFNETs), performs local checks on the availability of system resources. If adequate resources are available, the QoS manager establishes a local reservation state for the session. It also annotes the the datapath with a session handle for the session specific handling of data packets commensurate with their service requirements. Note that the data and the control paths are completely separate. This separation of control and data paths enables us to provide sophisticated control functions without sacrificing data path performance. Besides acting as the resource manager and the admission controller, the QoS Manager is also responsible for forwarding any network related control information, such as changes in reservation state, from the network to the application concerned. This is done via asynchronous messages posted to the application.

The QoS Manager and the supporting modules have been implemented on the IBM AIX platform. We have

[1] more specifically, those that support a sockets based communication system

extended the socket interface to provide an API to the QoS Manager. We have enhanced the memory allocator for session specific management of system buffers. The mbuf structure itself has been modified to act as the conduit for session specific information for efficient data handling. We have taken care to ensure that the mbuf modifications maintain object code level backward compatibility to accommodate third party network interfaces. We have also enhanced the architecture for the network interface layer and network device drivers for efficient packet classification and session specific packet handling. For example, we have modified the IFATM (network interface layer for classical IP over ATM) to establish separate ATM virtual channels (VCs) with appropriate QoS parameters for each RSVP session. We have also enhanced legacy LAN (token ring) drivers to support a service class based queuing structure.

The rest of this paper we briefly review design requirements that have guided our work and building blocks that comprise our QoS support architecture. The operational details of various components and performance results are reported in [1].

2 System Overview

Figure 1 shows an RSVP based quality of service architecture. In this example, S1 and S2 are sources, and D1,D2, and D3 are destinations of data. The sources, S1 and S2, as well as the destinations, D1,D2, and D3, run RSVP daemons that participate in RSVP protocol and exchange RSVP messages on behalf of their hosts. There are two basic types of RSVP messages – PATH and RESV. PATH messages are sent by the source and is associated with a data flow. PATH messages are encapsulated in IP or UDP datagrams. As PATH messages travel through the network towards the destination(s) they are intercepted by RSVP enabled IP routers on the path. The routers setup a soft state for the PATH messages they intercept. A PATH state block includes the previous and next hops of the flow and its traffic characteristics. When a PATH message reaches its intended receiver(s), it is processed by the RSVP daemon running there. If the receiver wants to make a reservation for the particular RSVP flow, it responds with a RESV message. The RESV message traverses the reverse path back to the sender. On the way to the sender, it is intercepted by RSVP enabled routers. If sufficient resources are available, a reservation soft state is established in the routers. Otherwise, a RESV ERROR message is issued and is sent back to the receiver. The RESV ERROR message is also intercepted by RSVP enabled routers and the reservation states are deleted. An end-to-end reservation is successfully established when the RESV message reaches the sender and is successfully processed by the RSVP daemon on the sender.

Reservation can also be made on a multicast session. In this case the sender sends PATH messages to a multicast group address. As in the case of unicast, the path messages travel through the network to all the members of the multicast group and PATH state blocks are established at all RSVP enabled routers in the multicast tree. When PATH messages reach the receivers, each receiver independently decides if it wants to request a reservation for the session. Each receiver can potentially request for different reservations for the same session. As the RESV messages from the receivers traverse upstream to the sender, they are merged by the routers at the merging points. Eventually, a reservation tree is established with the sender as the root and the receivers requesting reservations as the leaves.

In the example shown in Figure 1 D1,D2, and D3 are the members of the same multicast group, and S1 and S2 are the senders sending PATH messages to the multicast group address. As shown in the figure, each of D1,D2, and D3 receives two sets of PATH messages. The receiver D1 intends to make reservation on both the flows originating from S1 and S2, and sends RESV messages RESV1 and RESV2 in response to PATH messages PATH1 and PATH2, respectively. The receiver D2 want to make reservation only on the flow originating at S1 and sends RESV messages RESV1. The receiver D3 on the other hand decides not to make any reservation and does not send any RESV messages in response to the PATH messages from S1 and S2. The RESV messages are merged at the routers reservation trees are established as shown in the figure.

The classes of service to be supported in the Internet are currently under standardization. Two of the most important classes of service that are being considered are (1) guaranteed service, and (2) controlled loads service. Guaranteed service guarantees that datagrams will arrive within the guaranteed delivery time and will not be discarded due to queue overflows, provided the flow's traffic stays within its specified traffic parameters. This service is intended for applications which need firm guarantees on loss-less on time delivery of datagrams. Some of the interactive audio and video applications, and applications with hard real-time requirements fall in this category. The end-to-end behavior provided to an application by controlled load service closely approximates the behavior visible

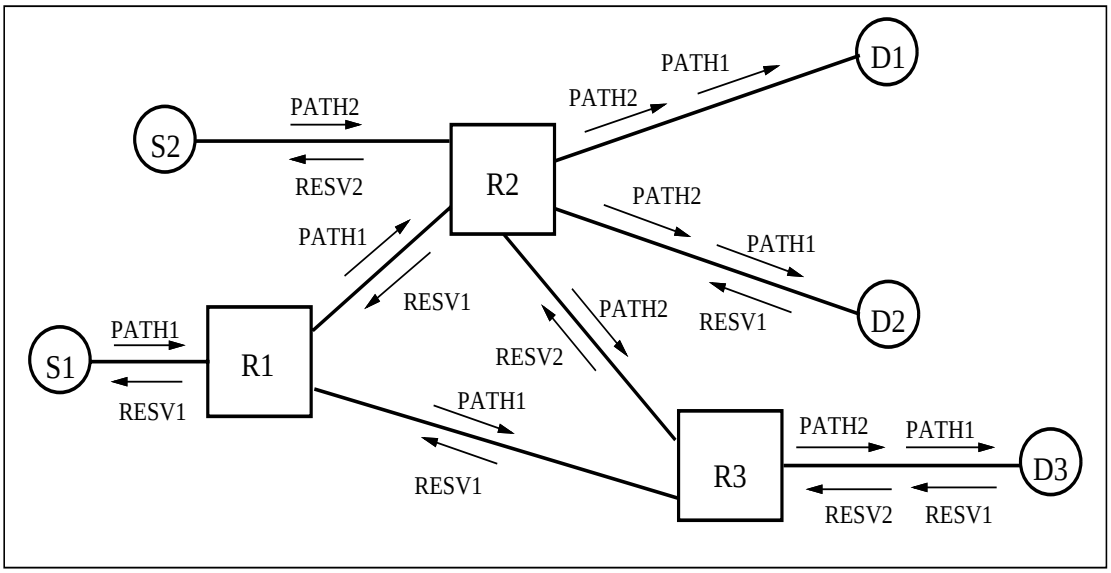

Figure 1: PATH and RESV message flows in RSVP.

to applications receiving best effort service under `unloaded` network conditions. That is to say, (1) a very high percentage of transmitted packets will be successfully delivered by the network to the receiving end-nodes, and (2) the transit delay experienced by a very high percentage of delivered packets will not greatly exceed the minimum transit delay experienced by any successfully delivered packet. Clearly, the definition of controlled load service is less precise than that of guaranteed service. It is intended for the broad class of applications which have been developed for use in today's Internet, but are sensitive to overload conditions. Some of the important members of this class are the adaptive real-time applications such as vic,vat,nevot etc.

3 Architectural Building Blocks

Figure 2 shows our software architecture of an RSVP enabled host. In this example, a number of applications are using RSVP signaling for resource reservation. The applications use the RSVP API (RAPI) library to communicate with the user level RSVP daemon running on the host. The RSVP daemon is responsible for translating the RAPI calls into RSVP signaling messages and local resource management function calls. For local resource management, the RSVP daemon interacts with the QoS Manager over an enhanced socket application programming interface.

The protocol stack in the kernel consists of a control plane and a data plane. The control plane is responsible for creating, managing, and removing reservations associated with different data flows. The data plane is involved in moving data from the application to the network and vice versa. The QoS Manager is the key component in the control plane of the protocol stack. It is entrusted with managing network related resources, such as network interface buffers and link bandwidth. It is also responsible for maintaining reservation states of different flows and the association between the flows and their reservations. Additionly, it performs traffic policing and shaping unless the network interface adapters perform these functions in hardware.

The QoS Manager is responsible for (1) allocating and managing network buffers, (2) policing and shaping of network bound traffic, and (2) maintaining reservation state of QoS connections. It is implemented as a separate protocol module and is accessed through the socket interface. Applications can access the services provided by the QoS Manager directly through the socket interface or via the RSVP daemon. For example, in order to set up new QoS connection, an application can use the RAPI interface to the RSVP daemon and communicate the end points of the connection and the traffic specification for the flow. The RSVP daemon uses the socket interface to communicate this information to the QoS Manager. It also prepares and sends appropriate RSVP signaling messages (PATH

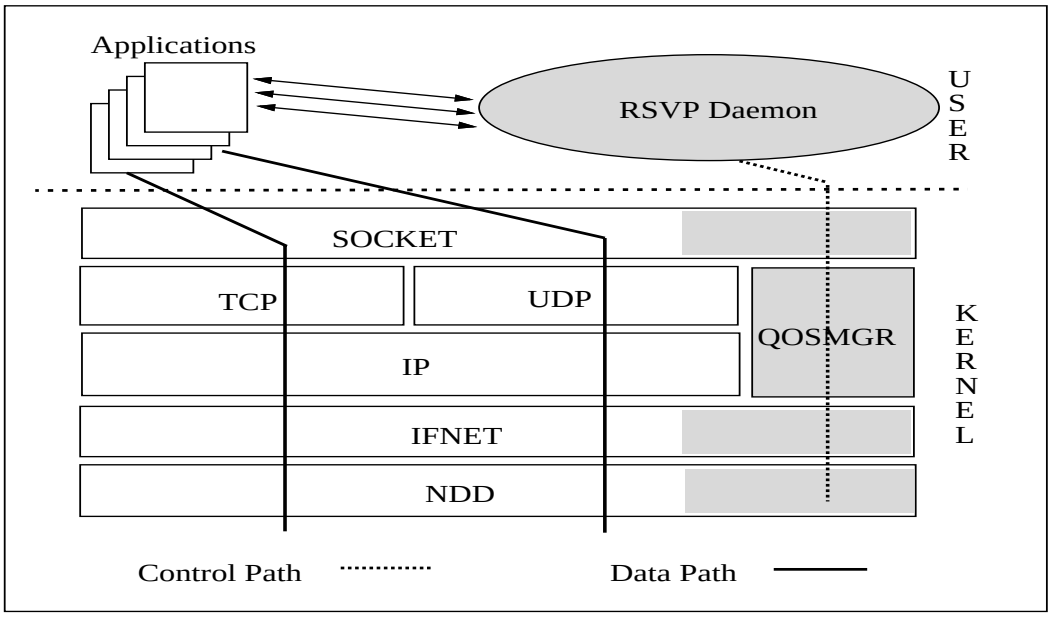

Figure 2: Protocol stack architecture and extensions.

and/or RESV) to the network. The QoS Manager sets up a reservation state for the connection. This include pre-allocating network interface buffers, initialization of reservation state, and performing admission control checks. It also annotates the data socket with the appropriate session handle. Similarly, the QoS Manager also gets involved when the application decides to modify the reservation level or to remove the reservation all together. It is also responsible for policing flows for Tspec compliance and blocking them when appropriate. In that sense it also has a part to play in the data plane.

The socket layer has been extended to support a new protocol family `PF_QOS`. Sockets with protocol family `PF_QOS` interfaces with the QoS Manager. `PF_QOS` sockets are also referred to as control sockets. Control sockets are used to avail services offered by the QoS Manager, specifically to create, modify, and delete reservations made on data connections. An alternative way of extending the socket interface would have been to extend the `getsockoption` and `setsockoption` kernel services. We chose to use the control socket mechanism over that of socket options because of its flexibility and architectural richness. Unlike socket options, control sockets interface can be used for (1) asynchronous upward control flow, (2) third party control on data flows, (3) sharing of reservation between multiple data sockets.

The network interface layer is responsible for implementing link-layer adaptation functions for different subnetwork types such as Ethernets, Token-ring, ATM, etc. We have extended this layer to provide local reservation services for a subnetwork. The local reservation services are provided as control path functions to higher layers (QoS Manager) through the I/O control (`ioctl`) interface. Although this control interface is common across all interface types, the capabilities of specific interfaces may differ substantially based on the characteristics of the network and the level of sophistication of the network interface device. The QoS Manager is cognizant of these levels of service and makes appropriate reservation and flow control decisions.

4 Summary

We have described the design and architecture of a framework for communication resource management for providing QoS support on Unix-like Internet servers, these being the typical source of multimedia data on the Internet. The heart of our architecture, which embraces emerging Internet standards for end-to-end resource reservations, is a new kernel module called QoS Manager. This module controls several important network-related resources, namely

bandwidth and transmission priorities on network interfaces and kernel buffer space (mbufs). We have also augmented the sockets layer to enable session specific handling of data packets. The QoS Manager and the sockets layer together provide a novel combination of buffer management and traffic shaping to provide a synchronous feedback mechanism for applications. These extensions preserve binary and API compatibility for sockets applications, while providing significant new functionality.

We have developed a prototype implementation of this architecture for the IBM AIX platform. We have implemented the QoS Manager and network interface support for ATM and token ring networks. This implementation is one of the first implementations of the RSVP protocol over an ATM network. When operating on an ATM network, our implementation provides QoS guarantees for TCP/UDP/IP applications with minimal increase in the pathlength, thereby achieving our goal for efficiency.

At the time of writing this paper, we have completed all aspects of the implementation for both the ATM and the Token Ring network interfaces. We are currently in the process of measuring and profiling the cost of providing QoS support for LAN interfaces. Also, we are integrating our implementation with the HTTP server to allow us to experiment with video and audio streaming over QoS-enabled RSVP connections.

References

[1] T. Barzilai, D. Kandlur, A. Mehra, D. Saha, and S. Wise. Design and Implementation of an RSVP Based Quality of Service Architecture for Integrated Services Internet. *IBM Research Report RC20618*, October 1996.

[2] M. Borden, E. Crawley, B. Davie, and S. Batsell. Integration of real-time services in an IP-ATM network architecture. *Request for Comments RFC 1821*, August 1995. Bay Networks, Bellcore, NRL.

[3] R. Braden, D. Clark, and S. Shenker. Integrated services in the Internet architecture: An overview. *Request for Comments RFC 1633*, July 1994. Xerox PARC.

[4] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) - version 1 functional specification. *Internet Draft draft-ietf-rsvp-spec-13.txt*, May 1996. ISI/PARC/USC.

[5] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Trans. Networking*, 3(4), August 1995.

[6] M. Perez, F. Liaw, A. Mankin, E. Hoffman, D. Grossman, and A. Malis. ATM signaling support for IP over ATM. *Request for Comments RFC 1755*, February 1995. ISI, Fore, Motoral Codex, Ascom Timeplex.

[7] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A new resource ReSerVation Protocol. *IEEE Network*, pages 8–18, September 1993.