Design and Implementation of an RSVP-based Quality of Service Architecture for Integrated Services Internet

Tsipora Barzilai†, Dilip Kandlur†, Ashish Mehra‡*, Debanjan Saha†, Steve Wise§

†IBM T. J. Watson Research Ctr.
30 Saw Mill River Road
Hawthorne, NY 10532
tsipora@watson.ibm.com
kandlur@watson.ibm.com
debanjan@watson.ibm.com

‡Real-Time Computing Lab
Department of EECS
The University of Michigan
Ann Arbor, MI 48109
ashish@eecs.umich.edu

§IBM Corporation
RISC System 6000 Division
11400 Burnet Road
Austin, TX 78758
swise@austin.ibm.com

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. 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 variety of network interfaces ranging from legacy LAN interfaces, such as Token Ring and Ethernet, to high-speed ATM interfaces. We also describe our initial experiences with the implementation of this architecture on the IBM AIX platform.

1 Introduction

As audio/video annotations become common features on Web pages and applications like InternetPhone, NetRadio, and WebTV become ubiquitous, the need for "better than best effort" Internet connectivity becomes inevitable. To address this, the Internet Engineering Task Force (IETF) is developing protocols and standards for Integrated Services on the Internet [4, 3, 6]. Applications request and reserve resources in the network and at the hosts using the Resource ReSerVation Protocol (RSVP) [17, 5]. Resource management is performed via per-flow traffic shaping and scheduling for various service classes [4] being standardized by the IETF, such as guaranteed [12] and controlled load [15]. Guaranteed service provides applications with a mathematically provable end–to–end delay bound using appropriate buffer and bandwidth reservation at all network elements. Controlled load service is designed for adaptive applications that do not need any specific quality of service, but can exploit the increased predictability in network performance.

To support integrated services on the Internet, network routers as well as end hosts must be enhanced to perform per-flow traffic classification, maintain flow-specific reservation soft states, and handle data 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 end hosts to enable RSVP-based quality of service (QoS) infrastructure in the Internet. We concentrate specifically on the design and implementation of QoS support on Unix-like (those supporting a sockets based communication system) Internet servers, the typical sources of multimedia data on the Internet.

One of the primary goals of this service architecture is to blend the QoS support with the existing TCP/IP stack and socket API, preserving the structure of the Unix networking subsystem. Applications not needing QoS support for communication should continue to run as is, and yet the QoS extensions should allow new applications to benefit from QoS enhanced network services. Further, control overheads for QoS support should not have any detrimental impact on data path throughput. Our design is also influenced by the observation that with the rapid penetration of the Web, potentially any Internet site can be a content provider, and hence a source of multimedia data. Thus, the design must scale from small to large number of connections and accommodate network interfaces with widely differing capabilities.

The heart of our QoS architecture is a new kernel module, the QoS Manager, entrusted with managing communication related system resources at end hosts. Applications or their designated agents request local resource reservations for a network session from the QoS Manager via an extended socket API. 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 resource availability checks. If adequate resources are available, it establishes local reserva-

*This work was performed while the author was visiting the IBM T.J. Watson Research Center.

0-8186-7813-5/97 $10.00

tion state for the session. It also annotates the datapath with a session handle for session-specific handling of data packets commensurate with their service requirements. Note that the data and the control paths are completely separate, enabling us to provide sophisticated control functions without sacrificing data path performance. Besides being the resource manager and admission controller, the QoS Manager also forwards network-related control information, such as changes in reservation state, from the network to the applications via asynchronous messages.

We have also augmented the data path from sockets to the network interfaces to enable session-specific handling of data packets. This is achieved through association of data sockets with session handles during reservation establishment, per-session pre-allocation and marking of kernel buffers (mbufs), and use of protocol-specific send routines to transfer data into mbufs. As the data packet traverses the protocol stack, the session handle carried in the buffer header is used to classify packets for session-specific handling. Similar enhancements to the receive data path are also possible. However, since our focus is on servers, and data transmission from a server dominates over data reception, we concentrate on the transmit part of the data path.

The QoS Manager and supporting modules have been implemented on the IBM AIX platform. Besides extending the socket API, 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 ensured that the mbuf modifications maintain object code backward compatibility to accommodate third party network interfaces. We have also enhanced the network interface layer and network device drivers for 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. Similarly, we have enhanced legacy Token Ring drivers to support service class based queuing.

Our work complements recent work on QoS-sensitive CPU scheduling of applications [14, 13, 8] and protocol processing [7, 10, 11] at end hosts. While these efforts focus on CPU scheduling, our primary focus is on the QoS support architecture exported to sockets based applications. With appropriate CPU scheduling support, our QoS architecture enables new and legacy applications to utilize end-to-end QoS on communication. We note that in many situations it may suffice to carefully assign execution priorities to application threads. The native-mode ATM stack described in [1] also performs traffic policing and shaping while copying application data into kernel buffers. However, our design is applicable to general TCP/IP protocol stacks, including legacy LAN and ATM interfaces, participating in an integrated services Internet. Moreover, the decision to shape traffic on a reservation is determined from the service class of the reservation.

In the rest of the paper, Section 2 gives a brief overview of integrated services on the Internet. We present the various building blocks comprising our QoS support architecture in Section 3. Section 4 describes our prototype implementation on AIX, and Section 5 presents preliminary performance results. Finally, section 6 concludes the paper.

2 System Overview

Below we present a brief overview of the RSVP protocol and the service classes under discussion in the IETF. A complete description of RSVP is provided in [5]. Details on different service classes can be found in [12, 15].

2.1 RSVP: An End-to-End View

Figure 1 shows an RSVP-based QoS architecture depicted data sources (S1, S2), destinations (D1, D2, D3) and IP routers (R1, R2, R3). The sources as well as the destinations run RSVP daemons that exchange RSVP messages (PATH and RESV) on behalf of their hosts. In general reservations can be made on multicast sessions, as depicted in Figure 1. In Figure 1, senders S1 and S2 send PATH messages to the multicast group address comprising D1, D2, and D3. The PATH messages travel through the network to all members of the multicast group and PATH state is established at all RSVP-enabled routers in the multicast tree; each of D1, D2, and D3 receives two sets of PATH messages. PATH messages arriving at their intended receiver(s) are processed by the RSVP daemon.

The receiver D1 intends to make (possibly different) reservations on the flows originating from S1 and S2, and sends RESV messages RESV1 and RESV2 in response to PATH1 and PATH2, respectively. The receiver D2 wants to make a reservation only on the flow originating at S1 and sends RESV message RESV1. The receiver D3 on the other hand decides not to make any reservations and does not send any RESV messages in response to the PATH messages from S1 and S2. As RESV messages from the receivers traverse upstream to the senders, they are intercepted by RSVP-enabled routers and if sufficient local resources are available, reservation soft state is established in the routers. The RESV messages are also merged at the appropriate merging points. An end-to-end reservation is successfully established when the RESV message reaches the sender and is successfully processed by the local RSVP daemon. Eventually, a reservation tree is established with the senders as the root and the receivers requesting reservations as the leaves. Note that PATH and RESV messages are independent of the data flow from the sender to the receivers although they follow the same route through the network. Hence, a reservation can be established before or any time after the data flow starts.

Additional details of refreshing PATH and RESV states and handling route changes are provided in [5]. An RSVP flow is uniquely identified by the five-tuple `<protocol,src address,src port,dst address,dst port>`. Filters are set up at routers and hosts to classify packets belonging to an RSVP flow and treat them in accordance with the reservation made

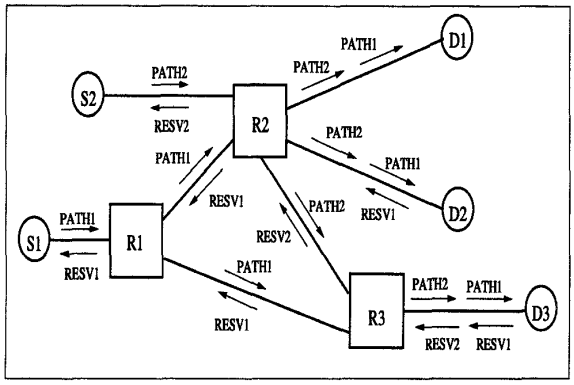

Figure 1. PATH and RESV messages in RSVP.

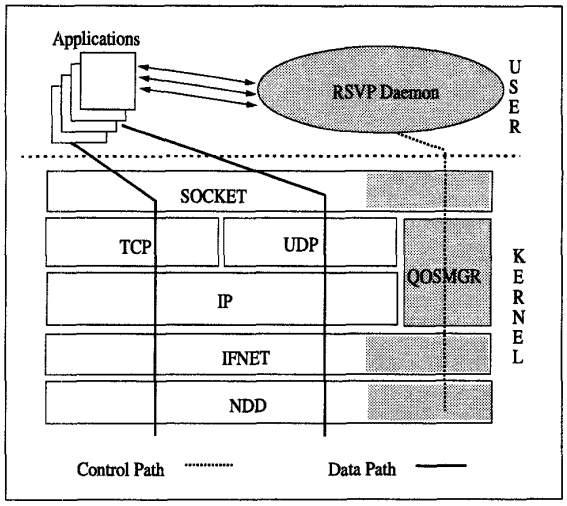

Figure 2. Protocol stack and QoS extensions.

on the flow. Note that, RSVP is just a signaling protocol that establishes reservation soft states at the end-hosts and routers. Honoring the reservations requires, among other things, resource and traffic management at the hosts and routers. The resource and traffic management mechanisms depend heavily on the service classes supported.

2.2 Service Classes

Two important service classes currently under standardization by IETF 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 that need firm guarantees on loss-less on-time datagram delivery. Some interactive audio/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 to applications receiving best effort service under `unloaded` network conditions. Controlled load service is intended for the broad class of adaptive real-time applications (such as vic, vat, nevot, etc.) developed for today's Internet that are sensitive to overload conditions.

To avail these services, a connection has to specify a traffic envelope, called Tspec, that is carried in the PATH message and includes a long term average rate, a short term peak rate, and the maximum size of a burst of data generated by the application. An application generating MPEG coded video could specify the average rate to be the long term data rate, peak rate to be the link bandwidth, and burst size to be the maximum size of a frame. Tspec also specifies the maximum and minimum packet sizes to be used by the application. For guaranteed service, traffic should be shaped to conform to the traffic specification. For controlled load traffic shaping at the source is not mandatory. Violating packets belonging to a controlled load session are allowed to pass the conformance check as best effort traffic. In addition to Tspec, guaranteed session specifies an Rspec containing the required rate of service and a slack term. The required rate of service should be at least as large as the long term average rate specified in the Tspec. The slack term signifies the difference between desired delay and delay obtained with the specified rate of service, and can be utilized by the network to reduce the reservation level of the flow. For controlled load service, there is no separate Rspec. Each controlled load session is guaranteed a rate of service equal to the long term mean rate specified in its Tspec. A session may receive better service if there is spare capacity in the system. We present next the design, implementation, and preliminary performance results from a prototype system that realizes an RSVP-based QoS architecture on end hosts. Refer to [2] for additional details.

3 Architectural Building Blocks

Figure 2 shows the software architecture of an RSVP enabled host. Applications use an 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 separate control and data planes The control plane is responsible for creating, managing, and removing reservations associated with different data flows. The data plane moves 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, and

is responsible for maintaining the reservation states of different flows and the association between flows and reservations. It also performs traffic policing and shaping unless the network interface adapters perform these functions in hardware. The data plane is an enhanced version of the classical Internet protocol suite. The extensions include changes to the socket layer, network interface layers, and network device drivers, for classification of network bound datagrams to the corresponding sessions and handling them in accordance with the reservation. Our prototype system realizes the architecture of Figure 2 on IBM RS/6000 servers running AIX (a 4.4 BSD variant) operating system.

Our design strives to achieve four fundamental objectives: (1) compatibility with Internet standards, (2) maintaining efficiency in data handling despite sophisticated control functions, (3) transparent integration of network interfaces of varying capabilities, and (4) ensuring backward compatibility with existing applications and system software. One of the skepticisms about QoS support in the Internet stems from the concern that it will affect data transfer performance. We have made careful architectural choices to maintain data path efficiency. Data path latencies of QoS connections in our system are the same and at times less than that of the regular data path.

Another important feature of our architecture is that it accommodates network interfaces of different types and capabilities, ranging from high function ATM interfaces (those with explicit support for QoS guarantees) to relatively low function (in terms of support for QoS guarantees) interfaces such as Ethernet and Token Ring. Our design aims to transparently accommodate the available functionality of the attached interface for a variety of LAN interfaces, and manage resources such as link bandwidth and buffers accordingly.

Implementing QoS extensions requires changes to the protocol stack, socket layer and network interface drivers. Fortunately, in most modern operating systems the network protocol stack and interface drivers are loadable kernel extensions. With careful design, it is possible to incorporate these extensions transparently into the kernel and other kernel extensions, as well as the applications using networking services. Our design strives to maintain backward object code compatibility with existing software as much as possible. Applications that do not use QoS support can also run as is. Our architecture also allows legacy applications (e.g., `ftp` and `telnet`) to utilize the QoS support provided without any modifications to application code.

3.1 QoS Manager

The QoS Manager plays an active role in the control as well as data planes. It is responsible for (i) maintaining reservation states of QoS connections, (ii) allocating and managing network buffers, and (iii) policing and shaping of network bound traffic. It is realized as a separate protocol module accessed through the socket interface, either directly by applications or by the RSVP daemon on behalf of applications. For example, in order to set up new QoS connection, an application communicates the end points of the connection and the traffic specification for the flow to the RSVP daemon via RAPI. The RSVP daemon uses the socket interface to communicate this information to the QoS Manager. The QoS Manager sets up reservation state for the connection, which includes pre-allocating network interface buffers, performing admission control checks, and annotating the data socket with the appropriate session handle. The QoS Manager is similarly involved when the application modifies the reservation or removes the reservation all together. In the data plane the QoS Manager polices flows for Tspec compliance, blocking them when appropriate.

Buffer Management: Each guaranteed and controlled load session is allocated a session specific buffer pool. When a new reservation is established, the QoS Manager uses the application Tspec to determine the number of buffers and size of each buffer required for the new reservation. The Tspec includes, the peak rate of traffic arrival r_p, the mean rate of traffic arrival r_m, and the maximum allowable burst size is b. It also includes the maximum and minimum packet sizes. The buffer pool should be large enough to hold at least a burst size worth of data including the packet headers. In general, it is possible to allocate buffers of any size that is a power of two. To keep the allocation simple, we populate the buffer pool with buffers of a single size, sufficient to hold maximum-sized packets. Our design assumes that applications perform path MTU (Maximum Transfer Unit) discovery and use that value for the maximum packet size so as to prevent fragmentation. The QoS Manager sets up the preallocated buffers as externally owned `mbufs` (kernel managed buffers) clusters, retaining the responsibility for freeing them at an appropriate time.

To permit shared reservations, the buffer chain is hung off the reservation structure associated with the QoS connection, and not associated directly with the data socket. Associating buffers with the reservation structure allows all `ftp` sockets to share the buffers, thereby sharing a single reservation for all `ftp` traffic. The free routine of each `mbuf` cluster is set via the `mbuf` extension header (added by us, as explained later) to allow reclaiming buffers from a reservation when it is released.

The QoS Manager provides an interface for the socket layer to request buffers. When an application sends data on a reserved connection, the socket layer requests the QoS Manager to allocate buffers from the reserved pool. Using the reservation handle presented by the socket layer, QoS Manager returns a buffer, if one is available, from the pool of preallocated buffers. If none is available, the application thread is either blocked or returned a best-effort buffer (if available) depending on the reservation type. For non-blocking sockets, the QoS Manager returns an error code under all circumstances where blocking would occur.

Traffic Policing and Shaping: All outgoing datagrams belonging to reserved connections are checked for Tspec compliance by the QoS Manager, which maintains two auxiliary variables, t_m and t_p, for each connection in its reservation state. Informally, t_m is used to check for conformance to the long term average rate specified in the Tspec. Similarly, t_p is used to enforce the short term peak rate. Both

t_m and t_p are set to $-\infty$ (a large negative number) at reservation setup. When an application makes a network send call with a data buffer of size L, t_m and t_p are recomputed as $t_m = max(t - b/r_m, t_m) + max(L, L_{min})/r_m$, and $t_p = max(t - L_{max}/r_p, t_p) + max(L, L_{min})/r_p$, where t is the arrival time of the packet, and L_{max} and L_{min} are the maximum and minimum packet sizes, respectively. Intuitively, t_m is the expected arrival time of the next packet assuming a rate of arrival of r_m and a burst size of b. Note that even if the application sends less than L_{min} sized packets it is charged for packets of size L_{min}. This is to discourage applications from sending small packets and increasing the fixed per-packet processing overhead at the host as well as in the network. Similarly, t_p is the expected arrival time of the next packet assuming a rate of arrival r_p and burst size of L_{max}, and helps maintain a minimum distance between consecutive packets in a flow. The application thread is in violation of its declared Tspec if $t < max(t_m, t_p)$. Tspec violations are handled differently depending on the type of reservation. In case of guaranteed service, the thread is blocked until it is conformant. Non-conformant control load connections can send excess traffic as best effort data.

Traffic policing and shaping is closely integrated with buffer allocation. The QoS Manager checks for Tspec compliance as well as availability of pre-allocated buffers. For guaranteed connections, the application thread is blocked if either of these checks fail; else, reserved buffers are returned from the pre-allocated buffer pool. For controlled load connections, if the checks fail the application thread is allocated best effort buffers, if available; else, the thread is blocked. On success reserved buffers are allocated from the pre-allocated buffer pool.

There are several advantages to this approach. One, performing traffic policing and shaping at a layer just below the socket layer makes it more efficient to block/wakeup application threads. The application need not worry about conforming to the traffic specification; the kernel blocks the write thread automatically as per buffer availability and Tspec compliance. Two, blocking the application write thread as soon as buffers become unavailable ensures that non-compliant data resides in the application's address space; this is consistent with the handling of buffer overflows in the socket layer in most Unix variant systems. Three, since the socket layer only copies as much as can be sent on the associated reservation, shared resources such as kernel buffers are kept available for other flows.

Maintaining Reservation States: The QoS Manager also manages reservation states for different flows. Multiple `control sockets` can be opened to the QoS Manager and used to pass control information. Multiple reservations can be created through a single control socket. The QoS Manager annotates the data socket for a reserved connection with the pointer to the associated reservation structure.

The QoS Manager also performs miscellaneous control functions, such as handling asynchronous notifications from the socket layer when data sockets are connected and disconnected. Since the control and data planes are completely separate, a reservation may be created before the data connection is actually established. Similarly, a data connection may be terminated without removing the associated reservation. The QoS Manager maintains the associations between sockets and reservations by registering handlers that are invoked by the socket layer in response to data socket connect and disconnect operations. It also keeps track of changes in connection state and notifies the owner of the control socket by posting messages via the socket interface.

3.2 Socket Layer Extensions

The socket layer extensions are in several dimensions. First, control sockets correspond to a new protocol family `PF_QOS`. The socket interface can also be extended via the `getsockoption` and `setsockoption` kernel services. We chose the control socket mechanism instead because of its flexibility and architectural richness. Unlike socket options, control sockets can be used for (i) asynchronous upward control flow, (ii) third party control on data flows, and (iii) sharing reservations between multiple data sockets.

Second, since QoS connections require traffic policing/shaping and buffer management, the socket layer must use the QoS Manager memory allocator for all sockets associated with QoS. Not only must it efficiently associate a data socket with a QoS session, it must also ensure that for all data transmissions on this socket, data is moved into pinned kernel buffers only when the buffer can be transmitted according to the associated QoS.

Third, we extended the protocol switch table to include protocol-specific send routines that are responsible for moving data from application space. For example, a TCP-specific send routine would move data only when it recognizes an opportunity for transmission. In contrast, most socket layer implementations move data into the kernel regardless of when it is sent out by the corresponding protocol. Buffer space is also consumed on a first-come-first-serve basis, which suffices for best-effort traffic but is unsuitable for QoS connections. Using QoS Manager for buffer allocation, in conjunction with the protocol-specific send routine, ensures that when the QoS Manager does return a buffer, it is highly likely that the packet is sent synchronously, thus allowing for reasonable traffic shaping.

3.3 Network Interface Layer

The network interface layer implements 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 through the I/O control (`ioctl`) interface. The interfaces may differ substantially in terms of network characteristics and sophistication of the network interface device. We have partitioned network interfaces into three broad categories:

NO_QOS: The interface does not support any reservation capabilities, e.g., unmodified legacy LAN interfaces.

STD_QOS: The interface supports admission control and scheduling but not traffic shaping. That is, the interface supports at least a priority queue mechanism capable of differentiating between the different service categories as shown

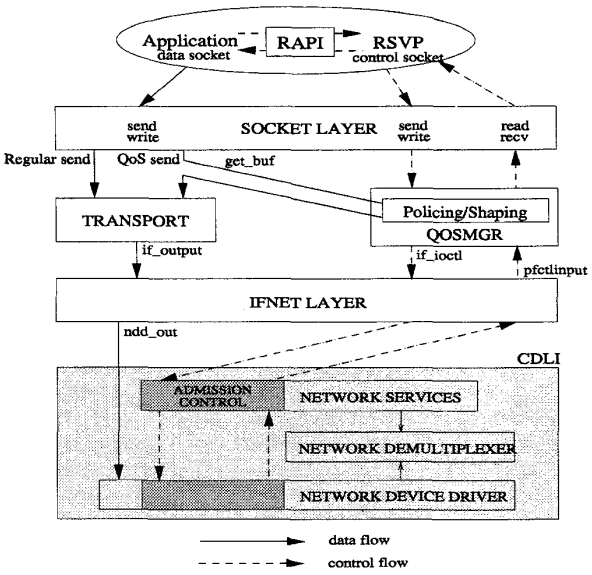

Figure 3. Call structure at host.

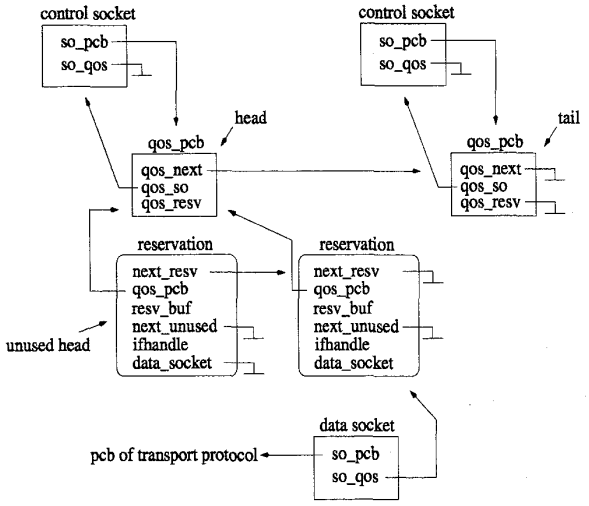

Figure 4. QoS Manager data structures.

in Fig. 5. The interface may also provide more sophisticated scheduling mechanisms within each service category.

HIGH_QOS: Besides admission control, the interface supports per-session traffic shaping, typically accomplished with hardware support such as the timing wheel shapers in ATM network interface devices.

The QoS Manager makes reservation and flow control decisions based on these service levels. No reservations can be made for sessions directed to a `NO_QOS` interface. For sessions directed to `STD_QOS` interfaces, the QoS Manager performs traffic policing and shaping, the latter being optional for certain service classes. A `HIGH_QOS` device absolves the QoS Manager from the potentially expensive operation of traffic shaping. Moreover, due to buffer preallocation, the flow control mechanisms continue to operate and an application is blocked due to lack of buffers when it tries to send data at a rate far exceeding its reserved rate. The network device drivers must also be extended appropriately.

4 Prototype Implementation

We have implemented this QoS architecture on RS/6000 based servers running AIX release 4.2 and equipped with ATM and IEEE 802.5 Token Ring adapters. Figure 3 illustrates the structuring of software layers and how they interact in response to RSVP and application requests.

4.1 QoS Manager

The QoS Manager maintains two kinds of associations: between control (`AF_QOS` domain) sockets and the reservations created via them, and between data sockets and the corresponding reservations. These associations must be kept relatively independent of one another since an application could close data and control sockets at different times. Closing a data socket should not affect established reservations, which are released either in response to an explicit application request or when the control socket is closed.

Figure 4 illustrates the data structures used by the QoS Manager. When a control socket (`qos_so`) is created, it is allocated a QoS protocol control block (`qos_pcb`) which points back to the control socket. The new `qos_pcb` is inserted into a list (pointed to by `head` and `tail`) of protocol control blocks, similar to the mechanism employed for other protocol control blocks. When a new reservation is established, a reservation structure is allocated and inserted into a list of reservations pointed to by the `qos_pcb`. The entire list of Internet protocol control blocks is searched to find a data socket that should be associated with this reservation. If such a socket exists, the address of the reservation structure is stored in a new `so_qos` field of the socket structure, and the reservation marked as used. This address is used during data transfer to efficiently identify the reservation structure to use for traffic policing and shaping on that data socket. The `so_qos` field is always null for control sockets.

If a connected data socket matching the reservation cannot be found, the newly-allocated reservation is marked unused and appended to the list of unused reservations pointed to by `unused head`. The QoS Manager receives socket connect/disconnect notifications from the socket layer via functions `qos_isconnected` and `qos_isdisconnected` registered by the QoS Manager when it is initialized. In response to a data socket connect notification, the QoS Manager searches the list of unused reservations for a matching reservation, and if one is found, stores the address in the `so_qos` field as before.

Session-specific handling of data packets is facilitated via a new QoS `mbuf` extension header with two new fields: the reservation handle associated with this packet, and a priority field indicating the service level this packet requires at

the network interface. For binary compatibility reasons, the QoS extension header is placed after the standard `mbuf` extension header. Note that with this design, only the control path incurs the overhead of associating data sockets with reservations. Since the reservation handle is passed in the `mbuf` extension header, there is no additional packet classification [16] overhead in the data path, keeping it efficient.

4.2 ATM Network Interface

The Turboways ATM adapter we use is a high-function interface supporting traffic shaping and scheduling in hardware. Consequently, no device driver extensions are required to support QoS connections on this interface. However, we extended the network interface layer (IFATM) so that RSVP sessions are not multiplexed on the virtual connection (VC) established for default IP, rather separate ATM connections are created for each RSVP session.

Creation of an RSVP session is initiated by the RSVP daemon on behalf of the application by making a reservation request to the QoS Manager. If enough buffers are available, QoS Manager passes the reservation request to IFATM using the I/O control (`if_ioctl`) interface, which we have extended with a new command (`IFIOCTL_QOS`) for reservation-related requests. If adequate resources are available, QoS Manager triggers a VC setup procedure for the RSVP session which involves the ATM call manager. The QoS Manager is notified of the success or failure of session establishment (VC setup) asynchronously via the `pfctlinput` upcall. Note that we can utilize the upcall mechanism to deliver asynchronously since the QoS Manager is implemented as a protocol module. Creation of separate VCs for different RSVP sessions requires modifications to the ARP (Address Resolution Protocol) table maintained by IFATM. In order to accommodate more than one connection to an IP destination, ARP entries to a particular destination are linked together and hooked off the IP destination entry in the ARP hash table. Deleting an existing RSVP session initiates tearing down of the VC serving the session.

On the data path, when a packet comes to IFATM it checks for the QoS header in the mbuf carrying the packet. If no QoS header is present, the ARP table is searched for the default IP VC handle to the packet's next hop destination. If the QoS header is present, IFATM extracts the VC handle from the QoS header. No ARP table search is required since this is the VC handle for the RSVP session to which the packet belongs. IFATM calls `ndd_output` with the VC handle to send the packet on the appropriate VC.

4.3 Token Ring Network Interface

We have also developed an implementation for the IEEE 802.5 Token Ring network adapter, which is an example of a `NO_QOS` interface. A similar approach can be used for Ethernet and other legacy LAN interfaces. The AIX token ring network device driver (NDD) does not provide any support for link bandwidth reservation. We have enhanced it from a `NO_QOS` level to a `STD_QOS` level by providing the traffic queueing and scheduling structure in Figure 5. These

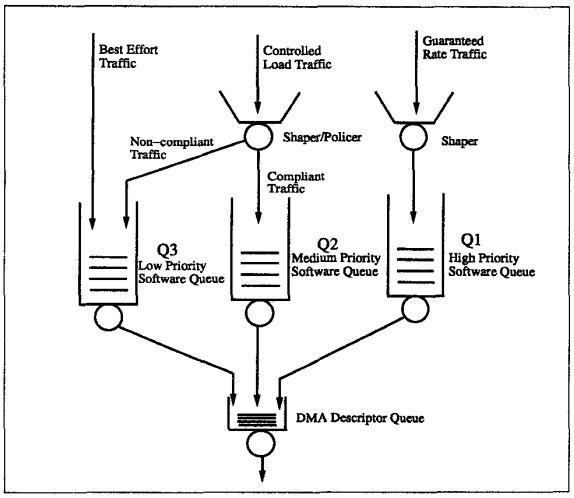

Figure 5. Shaping and scheduling of traffic.

enhancements are realized via extensions to the Network Services (NS) component of Common Data Link Interface (CDLI) [9] and the NDD (Figure 3). We do not address resource reservation on multi-access LANs, making our solution more appropriate for switched Token Ring or Ethernet.

Buffer management and traffic shaping is now provided by QoS Manager, as shown in Figure 3. However, the local admission control procedure during reservation establishment is split between the QoS Manager and our NS extensions. While the QoS Manager performs buffer requirement checks, NS (with possible support from NDD) performs link bandwidth checks. The control and data flow is same as with the ATM network interface, except that the `if_ioctl` notifications are now relayed to NS.

As shown in Figure 5, we enhanced the token ring NDD to support three packet queues, $Q1$, $Q2$, and $Q3$, where $Q1$ has non-preemptive transmission priority over $Q2$ and $Q3$, and $Q2$ has non-preemptive priority over $Q3$. Traffic from connections belonging to the guaranteed and committed rate classes is queued in $Q1$. Conformant traffic from controlled-load connections is queued in $Q2$. Non-conformant traffic from controlled-load connections and best-effort traffic is queued in $Q3$. The link scheduler services $Q1$ first; if there is a packet to transmit, it is dequeued from $Q1$ and prepared for transmission by inserting into the adapter's DMA descriptor queue. If $Q1$ is empty, $Q2$ is served; $Q3$ is served when both $Q1$ and $Q2$ are empty. Once the network adaptor is busy, the link scheduler is subsequently invoked in response to transmission completion interrupts.

The data path for packets destined for the Token Ring interface is the same as that the ATM interface, except that QoS Manager performs traffic shaping, if required. Another difference is that IFTR does not demultiplex packets based on the `ifhandle` carried in the QoS header. Demultiplexing of packets belonging to different sessions is done at the device driver using the priority field in the QoS header.

Machine	ADDRESV	DELRESV
vindhya	16.50ms	0.27ms
tista	27.00ms	3.40ms

Table 1. Control path latencies.

5 Performance Results

In this section we present performance results of some preliminary experiments using our prototype.

5.1 User Level Performance Profiling

First we measure user level performance of the control and data paths. These experiments exercise the various data paths through the QoS Manager. We extended the `netperf` program to create QoS sessions with different options for local traffic control such as policing, shaping, etc., and collect user-level statistics for packet transmission time. The tests were performed between two machines – (1) `tista` – RS/6000 model 530 with a 33.4 MHz POWER CPU, and (2) `vindhya` – RS/6000 model 42T with 120MHz PowerPC 604 CPU. Using the RS/6000 model 530 allows us to observe the impact of the new communication architecture on the vast existing base of older and slower servers. Both machines run AIX version 4.2 and are equipped with 100Mb/s Turboways100 ATM adapters connected via an IBM 8260 ATM switch. Although `tista` runs at a much lower clock speed, its overall performance is not proportionately worse than that of `vindhya`.

Table 1 shows user level latencies in creating and removing a reservation between the two machines. The latency involved in creating a reservation includes time taken to create local reservation states as well as signaling across the ATM network to setup a VC for the RSVP flow. The removal of a reservation includes cleaning up the local reservation state and tearing down the VC associated with the RSVP flow. However, the latency seen by an application in DELRESV does not include the time taken to tear down the VC. This explains the large difference between the latencies in ADDRESV and DELRESV. The most significant part of ADDRESV latency is contributed by ATM signaling overhead.

We also measure application-level UDP performance for different packet sizes and traffic control options. Table 2 shows the observed results with `vindhya` as the traffic source and `tista` as the sink. The first row shows the number of packets transmitted and received without any reservations (baseline). These packets traverse the best-effort path through the stack, and are carried on the best-effort ATM VC between the two systems (with a default setting of UBR with a 50 Mb/s peak transmission rate). The receiver is unable to cope with the unrestrained transmissions and is overrun.

The second row measures performance when a controlled load reservation is created with an average traffic rate corresponding to 1000 pkts/s (the rate varies with packet size when expressed in bytes/s). To illustrate the effects of traffic policing, the peak rate for the session was also set to correspond to 1000 pkts/s, while the bucket depth was set to correspond to 10 pkts. This results in the creation of a new QoS VC between `vindhya` and `tista` with the appropriate traffic parameters expressed in terms of ATM cells. Traffic policing was disabled in the QoS Manager and no packet-level policing was performed. Hence, the generated traffic was all directed to the QoS VC by the IFATM layer. The results show that ATM level traffic policing is performed on this session and packets in excess of the requested traffic rate are dropped at the source. The receiver receives packets at the smaller rate and is able to process them.

Connection Type		Message Size (bytes) 1400	2000	4000
Best Effort	Xmit	35294	27481	25526
	Recv	32	19	13713
Reserved	Xmit	45941	38985	29614
	Recv	9838	10027	9973
Reserved with Policing	Xmit	28306	22803	19174
	Recv	653	5494	6723
Reserved with Policing & Shaping	Xmit	10315	10243	10158
	Recv	9821	10025	9972

Table 2. Data path performance.

The third row measures performance with the same reservation but with traffic policing enabled in the QoS Manager. This corresponds to the recommended behavior for the controlled load service. The QoS Manager performs packet level policing for the session and transmits all excess traffic as best effort. The number of packets transmitted is thus higher, resulting in overruns at the receiver.

The fourth row measures performance for the reservation when both traffic policing and shaping are enabled in the QoS Manager. This is the recommended behavior for the guaranteed service, and may also be applied to the controlled load service. The QoS Manager performs packet level policing and blocks the application thread whenever it tries to transmit data in excess of its reservation. The number of packets transmitted now closely matches the reservation. The transmission rate is within the capabilities of the receiver, which receives most of the packets transmitted.

5.2 Kernel Overhead Measurements

We next present overhead measurements of the policing and shaping mechanisms (Table 3). Policing incurs the cost of retrieving the appropriate session state variables, computing the timestamps, and checking if a packet is compliant by comparing its expected arrival time with the current system time. Compliant packets are allocated reserved buffers from the pre-allocated buffer pool. Non-compliant packets are allocated best effort buffers from the shared `mbuf` pool. Buffer allocation from the shared pool is more expensive than buffer allocation from the session-specific reserved pool. This explains the difference between the policing overheads of compliant and non-compliant traffic.

Traffic shaping incurs additional overheads due to timer operations, blocking and wakeup of the violating threads.

Component Overheads		vindhya	tista
Policing	Compliant Traffic	24.0μs	44.0μs
	Non-compliant Traffic	36.0μs	87.0μs
Timer	Setting	7.4μs	14.0μs
	Handling	7.1μs	30.1μs
	Cancelling	6.5μs	9.6μs
Thread	Blocking	37.4μs	78.8μs
	Wakeup	14.4μs	23.8μs

Table 3. Overheads of different components.

Both policing and shaping increase the data path latencies for reserved connections over that of the best effort data path. Besides shaping and policing, the other differences between the best effort and reserved data paths are: (i) in the best effort data path system buffers are allocated from a shared buffer pool instead of a pre-allocated private pool of buffers as with reserved connections, and (ii) in the best effort path a search for the ARP entry is required; while in the reserved path the session handle maps directly to the appropriate virtual connection identifier. Both these differences reduce data path latencies on the reserved path and offset some of the overheads due to policing and shaping.

6 Summary

We have described the design and implementation of a QoS architecture for communication resource management in Unix-like Internet servers, these being the typical source of multimedia data on the Internet. Our architecture, which embraces emerging Internet standards for end-to-end resource reservations, is centered around a new kernel module called QoS Manager that manages network-related resources such as kernel buffer space and network interface bandwidth. We have also augmented the socket layer to enable session-specific handling of data packets. The QoS Manager and socket 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 while providing significant new functionality.

We also described our prototype implementation on the IBM AIX platform, for ATM and token ring networks. Ours is one of the first implementations of RSVP over an ATM network, and provides QoS guarantees for TCP/UDP/IP applications with minimal increase in path length, thereby achieving our goal for efficiency. Besides performing additional performance experiments, we are extending our prototype implementation with support for sub-network bandwidth management. We are also integrating our implementation with the HTTP server in order to experiment with audio/video streaming over QoS-enabled RSVP connections.

Acknowledgements: We gratefully acknowledge the contributions of John Chu and Isabella Chang at the IBM T. J. Watson Research Center, and Satya Sharma, Dan Badt, and William Hymas at IBM Austin.

References

[1] R. Ahuja, S. Keshav, and H. Saran. Design, implementation, and performance of a native mode ATM transport layer. In *Proc. IEEE INFOCOM*, pages 206–214, March 1996.

[2] 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. Technical Report RC 20617, IBM Research Division, November 1996.

[3] 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.

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

[5] 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.

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

[7] R. Gopalakrishnan and G. M. Parulkar. Bringing real-time scheduling theory and practice closer for multimedia computing. In *Proc. of ACM SIGMETRICS*, pages 1–12, May 1996.

[8] P. Goyal, X. Guo, and H. M. Vin. A hierarchical CPU scheduler for multimedia operating systems. In *Proc. 2nd OSDI Symposium*, pages 107–121, October 1996.

[9] G. Joyce. OSF NETSIG: Common Data Link Interface. *Design Specification*, April 1993.

[10] A. Mehra, A. Indiresan, and K. G. Shin. Resource management for real-time communication: Making theory meet practice. In *Proc. 2nd Real-Time Technology and Applications Symposium*, pages 130–138, June 1996.

[11] A. Mehra, A. Indiresan, and K. G. Shin. Structuring communication software for quality-of-service guarantees. In *Proc. 17th Real-Time Systems Symposium*, pages 144–154, December 1996.

[12] S. Shenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service. *Internet Draft draft-ietf-intserv-guaranteed-svc-05.txt*, June 1996. Xerox/BBN/IBM.

[13] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. K. Baruah, J. E. Gehrke, and C. G. Plaxton. A proportional share resource allocation algorithm for real-time time-shared systems. In *Proc. 17th Real-Time Systems Symposium*, pages 288–299, December 1996.

[14] C. Waldspurger. *Lottery and Stride Scheduling: Flexible Proportional-Share Resource Management.* PhD thesis, Technical Report, MIT/LCS/TR-667, Laboratory for CS, MIT, September 1995.

[15] J. Wroclawski. Specification of controlled-load network element service. *Internet Draft draft-ietf-intserv-ctrl-load-svc-03.txt*, June 1996. MIT.

[16] M. Yuhara, B. N. Bershad, C. Maeda, and J. E. B. Moss. Efficient packet demultiplexing for multiple endpoints and large messages. In *Proc. of Winter USENIX*, 1994.

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