Performance Analysis of An RSVP-Capable Router [1]

Tzi-cker Chiueh   Anindya Neogi

Computer Science Department
State University of New York at Stony Brook
Stony Brook, NY 11794-4400
{chiueh, neogi}@cs.sunysb.edu

Paul Stirpe

Reuters Information Technology, Inc.
88 Parkway Drive South
Hauppauge, NY 11788
paul.stirpe@reuters.com

Abstract

RSVP is a bandwidth reservation protocol that allows distributed real-time applications such as video-conferencing software to make bandwidth reservations over packet-switched networks. Coupled with real-time scheduling mechanisms built into packet routers, the network guarantees to provide the reserved bandwidth throughout the lifetime of the applications. Although guaranteed services are of great value to both end users and carrier providers, their performance cost, due to additional control and data processing overhead, can potentially have a negative impact on the packet throughput and latency of the RSVP-capable routers. The goal of this paper is to examine the performance cost of RSVP based on measurements from an industry-strength RSVP implementation on a commercial IP router. The focus is on the detailed evaluation of the performance implications of various architectural decisions in RSVP, as well as the effectiveness of RSVP in the presence of network faults. We found that RSVP's control messages do not incur significant overhead in terms of processing delay and bandwidth consumption. However, the performance overhead of real-time packet scheduling is noticeable in the presence of a large number of real-time connections. In extreme cases, the performance guarantees of existing real-time connections may not be kept, and some best-effort packets are actually dropped, although the overall bandwidth requirement from these connections is smaller than the available link bandwidth.

[1]The research was supported by a student grant from Reuters Information Technology, Inc.

1 Introduction

Distributed multimedia applications such as audio/video conferencing require end-to-end real-time performance guarantees from the underlying network. These applications can request guarantees on the quality of service in terms of bandwidth, delay, delay jitter, etc. The general approach towards real-time networking is that applications first make reservation requests on certain network service quality, and once the network determines that sufficient resources are available to accommodate such requests, it guarantees that service quality throughout the lifetime of the applications, by scheduling packets from these applications with higher priority inside network switches and routers. The priority levels are commensurate with the service quality requested. To ensure that applications do not abuse the service quality guarantee mechanism, a proper pricing mechanism is necessary to complete the real-time networking system.

Because network devices such as routers need to recognize packets from particular applications, they are required to maintain per-network-connection states that include information such as connection ID, resource reservation, and routing information. This is fundamentally different from traditional network architectures that never keep information about the applications at the end points. In terms of the OSI layering framework, the network strictly works at the *network* layer.

RSVP [ZHANG93] is a network resource reservation protocol that has been approved by IETF as an industry standard recently. RSVP was originally proposed as the reservation scheme for the Integrated Services Packet Networks architecture, which aims to integrate real-time and non-real-time network services within a unified framework. In its most general form, RSVP provides a robust mechanism for each network connection to leave per-connection state on the network devices along the path with which packets on that connection traverse the network. To keep it as generic as possible, the design of RSVP is intentionally decoupled from both resource reservation specifications and from routing. RSVP ensures delivery of the traffic specification and reservation objects without interpreting them. In addition, RSVP relies on unicast and multicast routing protocols to route the reservation messages, as well as the real-time data streams.

While various components of the Integrated Service network architecture have been developed over the last several years, most of the works in this area focused on either the development of the protocols or their impacts on the network system/applications through simulation-based studies. The actual performance overheads associated with the *implementations* of these protocols are largely left unaddressed in most cases. Two technology trends motivate us to carefully examine the performance cost associated with these advanced protocols, and investigate their scalability behavior with higher link speeds. First, the bit rates on both production LAN and WAN links are approaching Gbits/sec. Second, network devices are tasked with more and more functionality on a connection by connection basis. Both trends suggest that future network devices are required to carry out more and more work per unit time. Therefore, the feasibility of the whole reservation/priority-scheduling paradigm rests as much on the efficient implementations of the underlying protocols as on the protocols themselves. The goal of this paper is to report the measurement results of a detailed performance study on an industry-strength RSVP implementation on a state-of-the-art commercial IP router. To our knowledge, this is one of the first reports on RSVP's performance based on empirical measurements.

In Section 2, previous works related to this work are briefly reviewed. To set the stage for subsequent discussions on the performance measurements, a general description of the RSVP protocol and its architectural features that have performance implications are presented in Section 3. The experiment setup and evaluation methodology used in this work are described in Section 4. Section 5 includes the measurement results from the performance study and their analysis. Section 6 concludes this paper with a summary of the major findings and an outline of on-going work.

2 Related Work

To our knowledge, this is the first empirical measurement study of the RSVP protocol, with a focus on the overhead of supporting real-time network data transfers, and the performance implications of its architectural parameters. A simulation based study of the architectural decisions was made by Mitzel et al. [MITZ94] where the authors discussed how the design goals of RSVP overcome the shortcomings of STII [TOPO90]. [SCHW97] also discusses the applicability of RSVP but reports limited performance metrics to consolidate the claims. There have been suggestions regarding improvement of the soft-state timer management in [PAN97]. We study the timer mechanisms in detail and present the advantages and disadvantages of various alternatives. An alternative protocol called YESSIR [PAN98] has also been proposed recently. The goal is to design a light-weight reservation protocol with reduced number of messages, message types and protocol processing. But the flexible model of reservation as in RSVP, is lost in the process. The model makes some restricted assumptions such as sender-initiated reservation like STII, etc. In order to improve the scalability, the authors propose to use YESSIR in the local and regional networks and establish a small number of large-bandwidth "virtual paths" in the backbone, using RSVP.

3 RSVP Protocol Overview

RSVP is a transport layer bandwidth reservation protocol that uses out-of-band control messages to instantiate reservations and routing information on the intermediate network nodes. There are two types of control messages: *PATH* and *RESV*. These messages define QoS specifications based on IntServ formats. The formats of the message contents are opaque to RSVP and are interpreted by the traffic control modules at network nodes. The traffic control module consists of the *packet classifier*, which distinguishes packets that belong to different real-time sessions, the *packet scheduler*, which prioritizes the packets on an output link's waiting queue according to certain real-time packet scheduling discipline such as weighed fair queueing (WFQ), and admission control, which determines if there are sufficient resources available at the node to accommodate new requests.

Connection Establishment

RSVP is designed from the beginning to support multicast communication. Thus it chooses a *receiver-initiation* style for connection setup, similar in spirit to the IP multicast group management protocol (IGMP). This decision is also motivated by the requirement that heterogeneous receivers may request different QoS levels and thus should be charged differently for the same real-time session. When a unicast or multicast real-time connection is to be established, the real-time data source first sends the desired QoS request to the local RSVP daemon on the host. The host RSVP daemon then sends out a PATH message containing the session descriptor (source address and port number) and source traffic characteristics (**TSpec**: Token bucket rate, Token bucket size, max/min packet sizes) destined for the lone receiver or the multicast group. The first-hop router or the designated router on the sending host's Ethernet segment picks it up. From this point on, the PATH message travels through the network according to the unicast or multicast routing protocol implemented on the network nodes. RSVP is completely independent of the underlying routing protocol [2]. PATH messages are forwarded by each intermediate router to the next until the destination(s) subnetwork(s) is(are) reached. Along the way a PATH state is created in each node for this real-time session [3]. At the last hop, the router forwards the PATH message to the receiver host. In case of multicast there may be multiple receiver hosts on different LAN segments interested in the session. At each network element, the address of the last RSVP capable router is always extracted from the PATH message and kept in the PATH state block for the real-time session.

On receipt of a PATH message, the receiver requests the actual reservation in a RESV message, which is sent upstream to the last RSVP capable router. Contrary to PATH messages, RESV messages are always unicast even for multicast sessions. The RESV message contains a session descriptor, a style descriptor and a **flow-spec** object. The style descriptor specifies the reservation style, or which data sources can use the reserved resources. The possibilities are wildcard, fixed or shared-explicit reservation styles. If multiple sources can share a resource reservation, a wildcard filter may be used. For example, in an audio conference, normally one audio source is active at a time. For video traffic we may want to establish distinct reservations for different senders. This is a fixed filter reservation. A hybrid approach would be to partition senders into groups such that senders within a group are rarely active simultaneously, and to set up a shared reservation to satisfy their cumulative bandwidth requirement. Finally, the flow-spec object defines the QoS required by the receiver for the session. Each receiver can choose a different QoS for the same real-time session.

When the RESV message traverses towards the sender, at each node the QoS request is passed

[2] At each network node, the route query result may be cached however, to prevent repeated lookups. It is also possible that a particular interface has not been administratively enabled to carry reserved traffic. In that case the decision not to forward the message through that interface may be made by the RSVP daemon. Otherwise the IP layer needs to keep track of the protocol enabling information for each higher layer protocol and future ones that may be added.

[3] If PIM-SM is the multicast routing protocol then an additional delay may be needed for the routers to create the PATH state.

to the admission and policy control modules. The implementation of admission control is left to individual router vendors and is not specified. The policy control module checks on administrative permissions for making reservations. For example, certain nodes are only allowed to reserve certain amount of bandwidth. If both checks succeed, control parameters are set in the packet classifier and scheduler accordingly to ensure the desired QoS level when packets on the real-time session go through. On failure of any one of the checks, an error notification message is sent to the receiver that initiated the RESV message. The effective time to establish a reservation is the interval between the time when the first PATH message is sent from the source LAN segment to the time the first RESV message is received at the source LAN segment.

Soft State

Another architectural feature of RSVP is its use of *soft state* in adapting the resource reservations to changes in network routes and multicast group membership, and occasional node and link failures. The issue is that the reservation states installed on the network devices may need to be invalidated when the reservations are no longer required, such as when members leave and join the multicast group, or when the path used by a real-time data source to reach the receivers is no longer available. Instead of using explicit invalidations, RSVP chooses to treat the reservation states that will be automatically invalidated after a period of time unless being refreshed explicitly. This soft state mechanism simplifies the management of reserved resources in the face of changes because reserved resources can be re-cycled without being explicitly freed by the reserving nodes. However, during normal operation when there are no changes, both senders and receivers need to periodically initiate control messages to refresh the PATH and RESV states on the network devices. Essentially, RSVP optimizes for the changing case at the expense of additional control traffic in the no-change case. Because RSVP is designed to scale to large networks and large multicast groups, this architectural decision seems to be a good one.

For a network node, let R be the soft state refresh period with which its neighbor node sends in the control messages for a reserved connection, and let the node's local state lifetime for the corresponding connection be L. Each PATH/RESV message contains a TIME_VALUES object specifying the R value. To avoid message synchronization [FJ94], the actual refresh period used by the neighbor node is randomly set between $0.5R$ to $1.5R$ in most implementations. A node's local soft state timeout period for a reserved connection, L, is determined by the last received R for the corresponding connection, and a parameter K, which depends on the bit error rate of the link on which the control messages are sent. Specifically, K is a small integer that approximately represents the maximum number of consecutive packet losses that could possibly occur on the link. Given R and K, L is set to

$$L \geq (K + 0.5) \times 1.5 \times R$$

where in the worst case $K - 1$ messages may be lost without false state timeout. The value of R may be chosen locally by a node. R may be also changed dynamically to improve robustness.

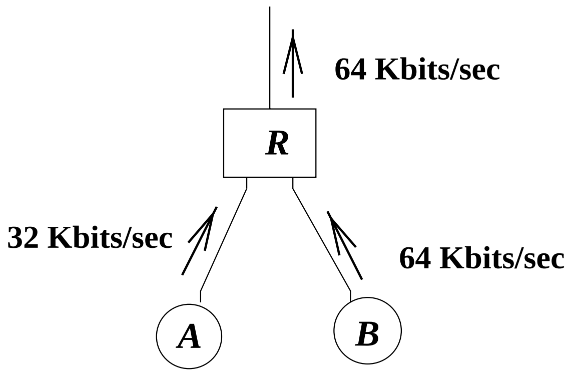

Figure 1: *An example network to illustrate the concept of Reservation merging.*

However R cannot be increased at an arbitrary rate. We will discuss the implications when we discuss soft state refresh overheads in Section 5.3, and fault recovery time in Section 5.5. Most implementations simply use a fixed timer value, i.e., R is set at 30 secs.

Reservation Merging

If a reservation state for a real-time session is already present at a network node, an incoming RESV message for the same session would not create a new reservation state. But the original reservation state may be updated to add a new reserved interface for the session. This process is called *reservation merging* and reduces the amount of resources reserved and the control message overhead substantially.

Figure 1 shows an example. A and B are two receivers requesting 32Kbps and 64Kbps reservations for the same real-time session. The router R will merge these two reservations and forward to the upstream node only the least upper bound (LUB) of these two RESV messages. If the data rate is beyond 32Kbps then only 32Kbps of the flow will be reserved for A and the rest will be delivered to A as best effort traffic. In addition, media filters may be installed in the routers to sensibly reduce the resource requirements of a real-time flow. For example a MPEG video flow may be reduced in quality to reduce bandwidth requirements for some receivers. The merging of reservation also makes it possible to limit the periodic refresh control message so that each link carries no more than a single path/reservation message in each direction during each refresh period for a given real-time session. In particular, each network node gets to independently choose the refresh interval with which to exchange control messages with neighboring nodes.

4 Evaluation Methodology

We use an IP router testbed installed in a commercial site for the performance experiments and traffic statistics collection. There are five routers in the testbed organized in a square-like topology. The network routers are Cisco 4700 routers each with 8 MBytes of DRAM as shared buffer memory and a 133MHz CPU as the control processor. The hosts are Pentium 200MHz machines with 64MB

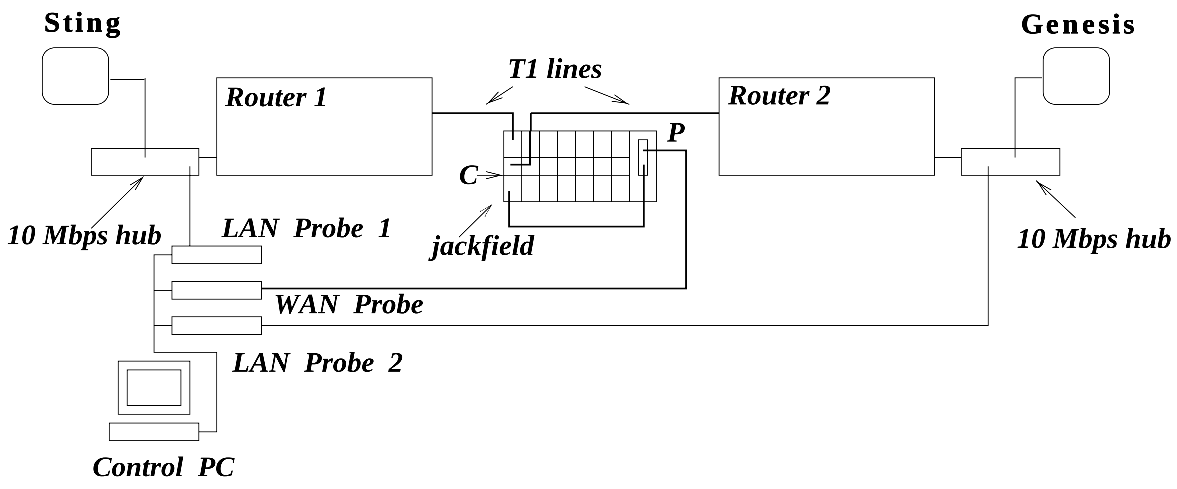

Figure 2: *The setup of LAN/WAN sniffers for throughput and latency measurements.*

memory. Both LAN and WAN links are available in the testbed. The WAN links run at 1 Mbits/sec and the LAN links are Ethernets at 10 Mbps and 100 Mbps. Packet traces were collected on the LAN and WAN segments, using 2 W&G LAN sniffers with 4MB memory and a W&G WAN sniffer with 24MB memory. We have to use network sniffers to accurately measure packet latency and system throughput because there are no measurement support facilities on the Cisco routers.

The sniffer probes are daisy chained and the first probe in the chain is connected to the parallel port of a PC. Thus all the sniffer probes are synchronized with the PC clock with negligible clock skews. The LAN sniffer probe can be connected to a port of the Ethernet hub. The WAN sniffer probe can be tapped into the WAN link using a jackfield, as shown in Figure 2. The serial line ports on the routers and the WAN sniffer both use a V.35 physical interface standard. The two router cables are plugged into the upper two jacks of a single column, Column **C** of the jackfield in Figure 2. The WAN sniffer probe is connected to one side of the double sided port P in the jackfield. A cable is then used to connect the other side of the port P with the third slot of column C in the jackfield.

The control PC is used to collect the trace files from the three sniffer probes, filter out irrelevant packets and save the traces on a hard disk as comma separated value (CSV) files. These trace files contain records separated by newline characters and fields separated by ','. Each trace record includes a packet's protocol headers and a timestamp. They can be read into any analysis tool which can handle CSV files. We have written our own scripts to filter relevant records and fields and analyze the traces.

The operating software on the routers is Cisco IOS version 11.2.9, which supports the Protocol-Independent Multicasting Dense-Mode (PIM-DM) protocol as the multicast routing protocol and Open Shortest Path First (OSPF) protocol as the unicast routing protocol. This seemed to be the most stable version as far as PIM-DM and RSVP are concerned. RSVP connections were set up using a RSVP Toolkit from Precept Software, Inc. This toolkit also includes a service application that can set up a real-time session by reserving bandwidth according to user-defined

QoS specifications. A large number of sessions were automatically set-up by capturing mouse and and keyboard events using Visual Test4.0. This software generates a Visual Basic script corresponding to the user events during the recording period while a single RSVP session is set up. The script may then be customized by hand, like introduction of a loop etc. to repeat the events for a specified number of times in order to set up a specified number of sessions. The Visual Basic script may also be written directly if one is proficient enough with the scripting language.

A major problem with the network sniffers is the limited trace length due to the small buffer memory. As a result, periodically the sniffers have to be stopped and the traces collected until that point are transferred to the control PC's disk for off-line analysis. Once the trace transfer is completed, a new cycle of trace collection and transfer repeats. Because the clocks on the sniffers are synchronized, it is possible to identify the "trajectory" of a particular packet by matching packet headers in the traces from adjacent network segments.

The following two trace snapshots were taken on the Ethernet and the serial line on Router 1 after filtering out other types of control messages.

```
Ethernet trace records :
-----------------------------------------------------------------------------
Abs Time        Destination         Source         Interpretation
-----------------------------------------------------------------------------
11:05:23.66931, 132.132.21.1,   sting.rrxnet,    RSVP Path Name=Session
Dest(IPv4)=GENESIS.rrxnet ID=UDP Dest_Port=9002 Src_Port=9002
11:05:23.68145, sting.rrxnet,   132.132.21.1,    RSVP Resv Name=Session
Dest(IPv4)=GENESIS.rrxnet ID=UDP Dest_Port=9002
11:05:53.67283, 132.132.21.1,   sting.rrxnet,    RSVP Path Name=Session
Dest(IPv4)=GENESIS.rrxnet ID=UDP Dest_Port=9003 Src_Port=9003
11:05:53.68561, sting.rrxnet,   132.132.21.1,    RSVP Resv Name=Session
Dest(IPv4)=GENESIS.rrxnet ID=UDP Dest_Port=9003
-----------------------------------------------------------------------------


Serial line trace records :
-----------------------------------------------------------------------------
Abs Time        Destination         Source         Interpretation
-----------------------------------------------------------------------------
11:05:23.67141, GENESIS.rrxnet, sting.rrxnet,    RSVP Path Name=Session
Dest(IPv4)=GENESIS.rrxnet ID=UDP Dest_Port=9002 Src_Port=9002
11:05:23.67834, 132.132.130.2,  router2.rrxnet,  RSVP Resv Name=Session
Dest(IPv4)=GENESIS.rrxnet ID=UDP Dest_Port=9002
11:05:53.67491, GENESIS.rrxnet, sting.rrxnet,    RSVP Path Name=Session
Dest(IPv4)=GENESIS.rrxnet ID=UDP Dest_Port=9003 Src_Port=9003
11:05:53.68179, 132.132.130.2,  router2.rrxnet,  RSVP Resv Name=Session
```

```
Dest(IPv4)=GENESIS.rrxnet ID=UDP Dest_Port=9003
----------------------------------------------------------------------
```

The reservation is from Sting to Genesis. Hence we see PATH messages with Source as Sting and Destination as Genesis on the serial line. On the Ethernet, PATHs are sent to the first hop router by the RSVP daemon on the host. Hence in the Ethernet trace records the PATHs have Destination set to 132.132.21.1, which is the Ethernet interface of Router 1. Since RESV messages are sent hop-by-hop, we see the Source is Router 2 and the destination is 132.132.130.2 (the next hop serial interface on Router 1) for the serial line trace. On the Ethernet trace the RESV messages are from the Ethernet interface 132.132.21.1 of Router 1 to the host. The trace records for different sessions are distinguished based on Dest_Port in this case.

On the control PC the traces collected from different sniffer probes are correlated to deduce such information as the latency of a packet through a router. As shown in Figure 2, a trace of an RSVP session may be collected, say between host1 and host2, using a chain of sniffers. Before the sessions are started, the sniffers are put in the capture mode. After a period of time, the sniffer capture buffers were dumped to the hard disk and analyzed off-line. To measure the delay for the first PATH message of the real-time session with destination UDP port 9002, through Router 1, we find the difference of the absolute timestamps associated with the packets (given by the sniffer) in the first Ethernet trace record and the first serial line trace record in the snapshots shown. A latency measurement is taken only for the first control message for the session. Therefore we can correlate the first messages for a given destination port from the two trace files. In our example the PATH message latency for port 9002 is $11:05:23.67141 - 11:05:23.66931 = 0.0021$ms.

5 Performance Measurements and Analysis

All performance measurements are on unicast reserved connections between the sender hosts, *Sting*, and the receiver host, *Genesis*, as shown in Figure 2. In some cases other hosts are used to generate network traffic through the routers.

5.1 Packet Latency

The packet latency through a router is measured by calculating the difference between the timestamps at which the packet appears on the input and output links. For RSVP control messages, only the first such message of a session is measured, since the routers do not immediately forward the subsequent control messages. Also, this measurement is useful to correlate with the connection setup time. Table 1 shows the latency of RSVP control packets in the loaded and unloaded cases when a certain number of real-time sessions already exist. Δ_P and Δ_R are the packet latency of a PATH and RESV message through a router. The first column indicates the number of real-time connections already existent before the connection being measured is established. The second and third columns show that the average RESV messages take more router processing than PATH messages, because the reservation-related processing such as admission control is actually performed

No. of Sessions	Unloaded		Loaded	
	Δ_P	Δ_R	Δ_P	Δ_R
0	2.00(1.00)	3.07(1.00)	4.30(1.00)	5.60(1.00)
9	2.82(1.41)	3.90(1.27)	5.98(1.39)	7.17(1.28)
99	3.90(1.95)	5.41(1.76)	8.51(1.98)	10.1(1.80)
499	4.93(2.47)	6.08(1.98)	10.79(2.51)	11.26(2.01)
999	5.44(2.72)	6.40(2.08)	12.04(2.80)	11.87(2.12)

Table 1: *Average Control Packet Latency (in msecs) when there are a variable number of real-time sessions, which may be loaded or unloaded.*

during the processing of RESV messages. The numbers in the parenthesis show the relative factor with respect to the case without any real-time session. In the unloaded case, there is no best-effort traffic and real-time sessions only generate RSVP control traffic but no actual data flow. Control messages for the real-time sessions are sent out using 30 seconds fixed refresh timers in this experiment. The performance impact of existing connections' control traffic on the setup time for new connections is minimal. For example, the control overhead of 999 connections only increases the packet delays of the first PATH and RESV messages of a new connection by a factor of 2.72 and 2.08, respectively.

To further isolate the router processing overhead due specifically to RSVP, we measure Δ_{IP_P} for the unloaded case, the average message latency of an IP packet of the same size as a PATH message, and Δ_{IP_R}, the average message latency of an IP packet of the same size as a RESV message in the unloaded case. In our testbed, Δ_{IP_P} and Δ_{IP_R} are measured to be 0.00140 and 0.00095 secs, respectively. Therefore the true RSVP PATH message processing overhead, $\Delta_{P_{header}}$, can be calculated as $\Delta_P - \Delta_{IP_P}$. Similarly, the RESV message processing overhead, $\Delta_{R_{header}}$, is equal to $\Delta_R - \Delta_{IP_R}$. Table 2 shows the pure CPU processing overhead associated with RSVP control packets. The 0-connection case demonstrates that the PATH message processing overhead is only half the data forwarding overhead, but the RESV message processing overhead is more than twice as much as the data forwarding overhead.

In the loaded case the same measurements were repeated with a constant load of 10K packets/sec with 64 bytes/packet pushed through the router by transferring packets between hosts on different Ethernet interfaces of the router. Figure 3 shows the network setup for this experiment. The control packet latency through the routers in this case is about doubled compared to the unloaded case. The relative latency increase is also slightly higher than the unloaded case because of non-linear queuing delay effects. But even in the worst case, the packet delay is low enough such that RSVP state timeouts will not occur unless consecutive messages are lost. It will be shown later that best effort packets start being dropped after about half the pipe is reserved for real-time connections. Thus PATH, RESV messages may also be dropped in such cases.

Table 3 gives an idea of the scalability of the host implementation of the router daemon with varying number of sessions handled by the host.

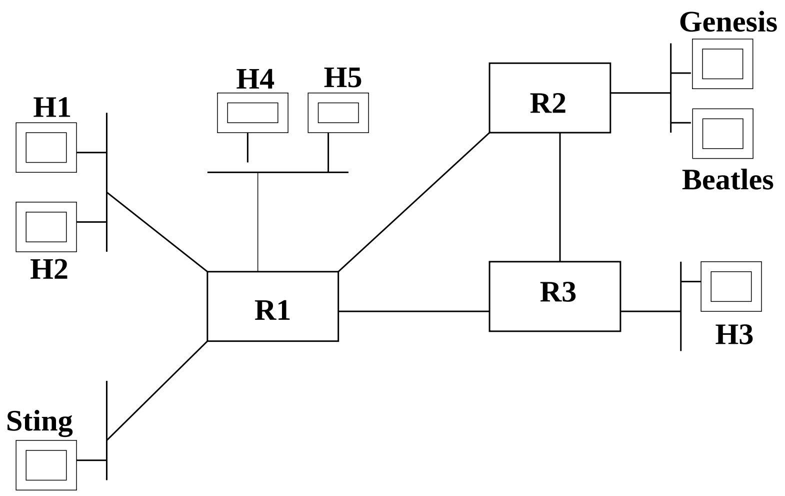

Figure 3: *The network topology of the measurement study in the loaded case. The router, R1, is stressed with traffic between H1,H2/H3, H4,H5, while the measurements are made on the connections between Sting and Genesis/Beatles.*

No. of Sessions	$\Delta_{P_{header}}$	$\Delta_{R_{header}}$
0	0.60	2.12
9	1.42	2.95
99	2.50	4.46
499	3.53	5.13
999	4.04	5.45

Table 2: *Processing overhead of RSVP header in routers (in msecs), when there are a variable number of real-time connections in the unloaded case.*

No. of Sessions	Δ_{PR}
0	1.72
9	1.85
99	2.14
499	2.56
999	2.98

Table 3: *Latency at receiver host(in msecs)*

No. of Sessions	M=2		M=3	
	Δ_T (model)	Δ_T (measured)	Δ_T (model)	Δ_T (measured)
0	11.86	12.00	16.93	16.84
9	15.16	15.14	21.88	21.91
99	20.34	20.20	29.65	29.52
499	23.74	23.60	34.75	34.81
999	25.40	25.60	37.24	37.39

Table 4: *Connection setup time (in msecs) from the model and the actual measurements when there are a variable number of unloaded real-time connections.*

5.2 Connection Setup Time

The connection setup time, Δ_T, is defined as the delay between the time when the first PATH message is seen on the sender's local Ethernet and the time when the first RESV message is seen on the same Ethernet. The receiver processing delay during connection setup, Δ_{PR}, is the latency between the time the first PATH message is seen on the receiver's local Ethernet and the time at which the first RESV message is seen on the same Ethernet. Let Δ_{T_n}, Δ_{P_n}, Δ_{R_n}, and Δ_{PR_n} be the corresponding Δ_T, Δ_P, Δ_R, and Δ_{PR} values when $n-1$ real-time sessions are already active. Therefore,

$$\Delta_{T_n} = M \times \Delta_{P_n} + M \times \Delta_{R_n} + \Delta_{PR_1} \tag{1}$$

where M is the number of routers between the sender and receiver, and we assume each of the n connections goes to a different destination host. Table 4 shows the measured connection setup time and the calculated connection setup time according to Equation 1 and the measurements in Table 1, when the real-time connections are unloaded. The number of hops, i.e., M, is 2 and 3 in the two cases shown. As can be seen, the numbers from the model and the measurement are very close. Similar measurements may be made to determine the approximate connection setup time when the network routers are under a given loaded condition. In this discussion, we ignore the propagation delay, which may be significant in large-scale wide-area networks.

The above discussion focuses on unicast real-time connection setup. In the case of multicast, the connection setup time should be similar to the unicast case, except that M should be the longest path between a sender and its farthest receiver in the multicast tree. The resulting estimate would be too conservative because merging of RESV messages should reduce the second term in Equation 1. For example, A RESV message from the farthest receiver may be dropped because an earlier reservation from another receiver that can accommodate it, is already made at some intermediate router.

5.3 Soft State Refresh Overheads

Most RSVP implementations use a fixed value for the refresh timer, R. Larger R values incur less control traffic, but result in longer recovery time when control messages triggered by route changes or network node/link failures are lost. To resolve this problem, a staged refresh timer scheme is proposed that adaptively increases the R value after a route change, starting from a small R value [BRDN96]. That is, when a RSVP-capable router notices a route change for a real-time connection, it starts to send out refresh control messages at R_1, R_2,...,R_s intervals, where $R_1 < R_2 ... < R_s$, and R_s is the stable soft state refresh interval. This method makes it possible to use large R_s values to reduce the control overhead during the stable period without lengthening the fault recovery time. Another minor disadvantage of large refresh timer values is that the period during which the resource reserved by a real-time connection is wasted after the connection is torn down, is longer because some connections can only be torn down implicitly through soft state timeouts. The Cisco routers in our testbed implement a trivial version of staged refresh timers. After a network failure or connection startup, control messages are immediately forwarded, and from the next message onward, R is set to about 30 secs. This is why all our measurements of control packet latencies had to be taken only for the first message for a session.

If R_f is the fixed refresh timer value, then the bandwidth overhead for control traffic with the respect to the total available link bandwidth is

$$\frac{\frac{1}{R_f} \times (PATH_{size} + RESV_{size})}{Bandwidth}$$

where $Bandwidth$ is the link bandwidth. Figure 5 shows the bandwidth overhead for a fixed refresh scheme for various line capacities and a fixed refresh interval of 30 secs. In the case of staged refresh timers, let R_l be the refresh interval, R for the first refresh message. Let the constant rate of increase of the refresh interval be r, i.e. R_i/R_{i-1} is r. Let the R_s be the stable state refresh timer value, as already defined. Finally, let t be the minimum of mean time between failures and expected session duration time.

The number of messages required to reach R_s from R_l with a rate of increase r is $\log_r(R_s/R_l)$. The time taken to reach R_s from R_l with a rate of increase r is defined as t_d, which is equivalent to $R_l + rR_l + r^2R_l ... + r^{\log_r(R_s/R_l)}R_l$, or $R_l(r^{\log_r(R_s/R_l)\ +\ 1} - 1)/(r-1)$.

If $t > t_d$, as shown in Figure 4, then the control bandwidth overhead is

$$\frac{(\log_r(\frac{R_s}{R_l}) + \frac{t-t_d}{R_s}) \times (PATH_{size} + RESV_{size})}{Bandwidth \times t}$$

If $t < t_d$, then the number of messages sent during t is x, where $R_l + rR_l + r^2R_l .... + r^xR_l \ = t$, and therefore $x \ = \ \log_r(\frac{t(r-1)}{R_l} + 1) \ - \ 1$. Consequently, the control bandwidth overhead is

$$\frac{\log_r(\frac{t(r-1)}{R_l} + 1) \ - \ 1) \times (PATH_{size} + RESV_{size})}{Bandwidth \times t}$$

The value of r cannot be arbitrary because if one or more successive control packets are lost, the receiving node's timer value may be out of date, and a false state timeout with state deletion may result [BRDN95].

To further reduce the refresh message overhead, an acknowledgement-based control messaging scheme may be used [PAN97]. An *echo-reply* message is used to explicitly acknowledge each received

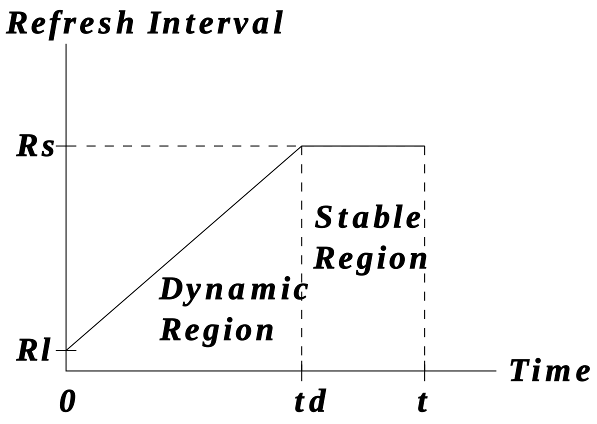

Figure 4: *The evolution of the timer value in the staged-timer scheme within the lifetime of a RSVP session.*

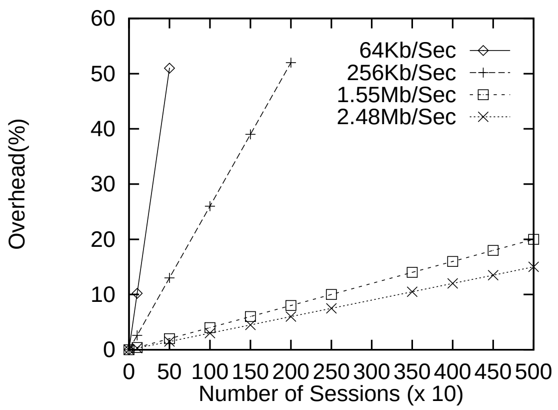

Figure 5: *The Bandwidth overhead with varying number of sessions for various line capacities, if a fixed refresh timer with refresh interval = 30 secs is used.*

control message. As soon as the sender node receives the acknowledgement, it can immediately switch to the stable state refresh interval, R_s. The receiving node should now expect a refresh interval of R_s and tunes its state life timer accordingly. Changing the state life timer is critical in this case because otherwise it may lead to pre-mature soft state timeouts and deletion. The only problem with this *echo-reply* scheme is that it requires a new message.

5.4 Throughput Impacts of Real-Time Packet Scheduling

Once the reservation states have been established in the routers, some sort of real-time packet schedulers are required to schedule the packets in the transmission queue associated with each output link to ensure that the real-time performance guarantees are indeed met at run time. Compared to conventional IP packet forwarding, real-time packet schedulers incur additional performance overheads. In our testbed, Cisco routers implement some variant of packetized weighted fair queuing, which incurs a well-known $\log_2(K)$ overhead of sorting the first packets of the K currently active sessions according to the priority indices. When there are only non-real-time sessions, the packet scheduler simply uses the FIFO policy and therefore doesn't cause extra overheads. In this subsection, we quantify these performance costs under different workload conditions.

Each output interface of real-time routers typically could be configured to grant a fraction of the link bandwidth to real-time flows and leave the rest for best-effort traffic. Based on the measurements on the packet loss rate and increased latency for best-effort traffic, one can determine what the maximum percentage of the link bandwidth should be reserved for guaranteed flows.

Figure 3 shows the network setup for the following measurements. Only *Sting*, *Genesis* and *Beatles* are used as hosts for traffic generation or reception. Guaranteed flows are set up from *Sting* to *Genesis* and best-effort connections from *Sting* to *Beatles*. First, a best-effort session is set up with constant packet injection rate of 1K packets/sec with 64 bytes per packet. Thus half the link bandwidth, 1 Mbits/sec, is filled with best-effort traffic and the router R1 is fed with constant load throughout the experiment. Then a different number of real-time sessions are established, each with a bandwidth reservation of 125 bytes/sec. The average latency of 100 best-effort packets is calculated from the traces collected from the links *Sting-R1* and *R2-Beatles*, and on the link *R1-R2*. Next we replace those real-time sessions with one best-effort session that consumes the same amount of bandwidth as those real-time sessions as a whole. Again the average best-effort packet latency is measured. In both cases, packet loss rates for best-effort traffic are also calculated from the number of best-effort packets sent out from **Sting** and the number of packets received at **Beatles**.

The experiment is repeated for an increasing number of real-time sessions, and its best-effort counterpart. The best-effort session which fills half the pipe is constant throughout all these measurements. The total traffic through the router never exceeds the link bandwidth, 1 Mbits/sec. Packets for real-time sessions are generated on a cycle by cycle basis. In every cycle a random order is chosen for the packets belonging to different real-time sessions. This random ordering is necessary to stress the real-time packet scheduler inside the router. In every cycle, the best-effort packets are sent after all the the real-time sessions' packets. This ensures that best-effort traffic never experiences queuing delay that is due specifically to low scheduling priority.

The difference in the best-effort packet latency measurements for the case when a fixed number of real-time sessions go through the router, and for the case when those real-time sessions is replaced with a best-effort session with equal aggregate bandwidth, represents the performance overhead due to the router's real-time packet scheduler. Table 5 shows the measurements of the packet latency and packet loss rate under different numbers of real-time sessions. The second column shows the percentage of the link bandwidth consumed by real-time connections. The third and fourth columns show the best-effort packet latency under real-time and best-effort loads with the same bandwidth requirement. The difference between the two, as shown in the fifth column, is the real-time packet scheduler overhead. The sixth and seventh columns show the best-effort packet loss percentages under real-time and best-effort loads.

No. of RT Sessions	**% of Bandwidth**	**Pkt Latency with RT**	**Pkt Latency with NRT**	**Scheduler Overhead**	**Pkt Loss with RT**	**Pkt Loss with NRT**
1	0.1	0.51	0.46	0.05	0%	0%
10	1	0.54	0.47	0.07	0%	0%
100	10	0.59	0.49	0.10	0%	0%
400	40	1.65	1.52	0.13	0%	0%
450	45	3.87	2.59	1.28	1%	0%
500	50	4.11	2.71	1.40	6%	0%
1000	100	x	2.98	x	100%	0%

Table 5: *The best-effort packet latency and packet loss rates under real-time and non-real-time loads of the same aggregate bandwidth requirement. All latency values are in milliseconds*

The real time scheduling overhead increases with the number of real-time sessions, because the priority sorting delay depends on the number of active connections. As shown in Table 5, after about 400 real-time sessions, the real-time scheduling overhead seems to increase drastically. This increase is mainly due to the fact that the packet scheduler cannot keep up with the packet arrival rate, and as a result, each real-time packet will incur extra queuing delay in addition to the scheduling delay. To verify that the bandwidth reservations indeed start to break down at this point, we measured the amount of bandwidth that the router gives to each real-time connection. The average bandwidth given to a sample of sessions was measured over a period of around 30 minutes. Table 6 shows that after 400 sessions the packet scheduler overhead is too much for the router to be able to keep up with these many real-time flows.

The best-effort packet delays and loss rates in Table 5 can also be used to determine the number of real-time sessions that should be admitted without impacting the best-effort traffic. For example, at 450 real-time sessions, although the total amount of bandwidth required is less than the total

No. of Sessions	Sample1	Sample2	Sample3	Sample4	Sample5
1	125.8	NA	NA	NA	NA
10	125.1	126.3	125.3	126.1	125.2
100	125.9	125.2	125.1	125.5	124.7
400	125.1	124.2	126.8	125.3	124.8
450	121.1	119.8	121.2	121.6	118.6
500	119.2	124.1	115.9	120.1	114.2
1000	110.1	115.2	112.1	112.2	119.9

Table 6: *Sampled measurements of the received bandwidths in the presence of a varying number of real-time connections. The original reservation is 125 bytes/sec.*

link bandwidth, the best-effort packet loss rate is 1%. This indicates that the real-time packet scheduling overhead when the number of sessions is large can decrease the effective link bandwidth, and therefore causing best-effort packet dropping.

Because RSVP control messages are essentially best-effort packets, congestion may cause consecutive control packets to be dropped, leading to false soft state timeout. It is thus questionable whether control messages should be sent as best-effort packets, because performance guarantee is most needed during congestion, but during congestion the control messages for resource guarantees actually have a high probability of getting dropped. One possible solution may be to set the TOS field inside the IP packets to assign a higher priority to the control messages, i.e., in-band-signalling, and require RSVP-capable routers to treat these packets specially during packet forwarding.

5.5 Fault Recovery Time

The fault recovery time of a RSVP capable network may be decreased by reducing the refresh interval. We have already explained how staged refresh timers can reduce the failure recovery time. In a multicast session, if one of the links or interfaces on a router fails, then the corresponding new interface should switch to a high refresh frequency, but the other interfaces should continue with the stable refresh frequency. The router should not flood the stable regions of the multicast tree with unnecessary refresh messages, just because one link on the multicast tree fails. For this to work, separate refresh timers need to be maintained for different interfaces of a router for the same session. In a fixed refresh timer scheme, if the first refresh message after fail-over is lost, we have to wait an entire refresh interval for a recovery procedure to continue. In the case of a staged refresh timer however, the failure recovery time is largely insensitive to the refresh interval in the stable case, R_s. A large number of messages have to be lost in succession to make the recovery time sensitive to R_s.

Even if staged refresh timer is implemented, the major portion of failure recovery time is the convergence time of the routing protocol. The RSVP implementation we have, waits about 2 secs for the routing protocol to settle. During this time all reserved traffic previously flowing through the failed node or link are given best effort service. This is unacceptable in most real time flows

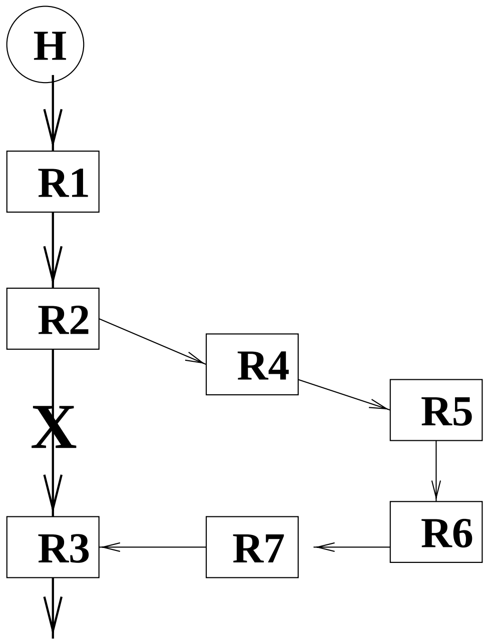

Figure 6: *Re-establish reservations via alternate routes during network link/node failures. In this case, the link between R2 and R3 is down.*

that require hard guarantees of the QoS even during the fail-over period.

Let us take the unicast routing case as in Figure 6. If the link between R2 and R3 fails, the routing protocol, say OSPF, will establish an alternate route through R2-R4-R5-R6-R7-R3. Let, Δ_{OSPF} be the time taken by OSPF to converge to a new route, $\Delta_{PR_{router}}$ be time taken by a router to send a RESV message upstream after it receives a PATH message from a different previous hop, and Δ_G be the time taken to re-establish the reservation after failure, i.e. the gap in reserved service. Δ_{OSPF} is a function of the length of the alternate path. We have not measured OSPF convergence times on our testbed because it is difficult to extrapolate measurements of Δ_{OSPF} obtained from the small testbed for arbitrarily large networks. However, we assume a good network design should ensure that the route convergence time after a link/node failure should be relatively small with a relative large network.

After the routing protocol converges, a reservation has to be established along this alternate route. The reservation latency on this new segment is purely a function of the length of this alternate route. When a router receives a PATH message from a previous hop different from the current one in the PATH state block [ZHANG96], it immediately sends a RESV refresh to the new previous hop, which is then forwarded upstream. In this example R3 will receive a PATH message from R7 since R2-R3 link has failed.

If N is the number of routers on the alternate route, then

$$\Delta_G = \Delta_{OSPF} + N \times \Delta_P + N \times \Delta_R + \Delta_{PR_{router}}$$

As in connection set-up times, we ignore the propagation delay, which may be significant in large-scale wide-area networks. Here we have assumed that when the local OSPF module notifies RSVP about a local interface or link failure, RSVP will send out a refresh on the new interfaces for those sessions that have been re-routed, immediately after waiting for the OSPF to converge. Also, we assume that there is indeed a new route and the routers on that route have sufficient resources to admit the re-routed flow. If that is not the case, the connection has to be torn down. Assume $\Delta_{PR_{router}}$ is of the same order as Δ_{PR} measured on the host (3ms for 1000 sessions, as shown in Table 3), if N is around 10, then by substituting Δ_P and Δ_R with the measurements (12ms for 1000 sessions, as shown in Table 1), we get $\Delta_G = 2 + 0.243 = 2.243$ secs.

However, for certain applications, the fault recovery time, Δ_G, must be reduced to preferably zero. There are two separate issues that need to be addressed to support fault-tolerant real-time networks. First, a new route from the source of each connection inflicted by a network failure to its destination(s), that is not affected by the network link/node failure and that has sufficient resources must be identified quickly. Secondly, the same reservation must be made along the new route to re-establish each affected connection.

Because the route convergence time is the major cost, one method to reduce the fault recovery time is to pre-compute a secondary route between pairs of consecutive nodes on the primary path and perhaps to make the same reservation on the secondary route in advance. When a link fails, the end points of this link can decide to switch all the connections previously travelling over the failed link to the secondary path immediately. As a result, the fault recovery time is close to 0. A reservation is established along the alternate

Of course, the disadvantage of this approach is that it is too expensive to have alternate path reservations among all pairs of consecutive routers. A more scalable and less expensive option would be to establish these duplicate reservations only among some of the *stable routers* on the primary path. In this case, the primary and secondary paths must be disjoint. These routers are stable in the sense that their software/hardware are fully replicated and assumed to never die. When some link on the path between a pair of stable routers fails, a special message should be sent to the upstream stable router, which would then switch all real-time connections affected by the failed link to the alternative path. The fault recovery time in this case is not zero, but is bounded by the number of hops between consecutive stable routers. Given the fault recovery time measurement and the application requirement, the network designer can structure the network topology and position stable routers accordingly.

6 Conclusion

Several important conclusions can be drawn from this performance study of RSVP-capable routers. First, the control message latency of RSVP is reasonable compared to normal IP forwarding, and the control message bandwidth overhead can be carefully controlled without any serious negative side effects. Second, it is a good idea to send RSVP's control messages at higher priorities. Admittedly, most existing IP routers ignore the priority field of the IPv4 headers. However, since RSVP requires modification to the IP routers anyway, it may be worthwhile to include support for prioritization. Third, the performance overhead associated with real-time packet scheduling at each output port is non-negligible. The admission control module should factor this overhead in the admission decision-making process given the existing real-time connections. Finally, although RSVP is designed to be routing protocol independent, this decision makes it difficult to implement efficient connection setups because the routing metrics used by existing routing protocols do not necessarily reflect what RSVP wants. It also creates problems for supporting fault tolerance using redundant path reservation, because most routing protocols only support single-path rather than multi-path routing. Of course, the measurements reported in this paper are performed on a relatively small testbed with low-end routers. All the measurements should thus be interpreted with respect to such a setup. However, we believe the methodology we used to carry out this study is equally applicable to other testbeds, and may help others to perform similar studies for different configurations, thus advancing the field's understanding of the performance behavior of RSVP.

We are currently working on the following issues related to this project. First, we are looking at extensions of RSVP that can provide a general framework to trade off between amount of redundant reservations and fault recovery time. Second, we are planning to build a larger RSVP network testbed using commodity PCs to experiment with a larger number of hops and faster link speeds. Thirdly, we are currently implementing RSVP under the OPNET simulation system, and plan to port it over Linux. Finally, we are investigating a scalable weighted fair queuing technique that can scale up with higher link speed while providing a guarantee bound that is close to the fluid fair queuing model.

References

[BRDN95] Braden R., Estrin E., Steven B., Herzog S., Zappala D.; *The Design of the RSVP Protocol*; USC/ISI FINAL REPORT, Contract DABT63-91-C-0001 27 May 1993-30June 1995

[BRDN96] Braden R., Zhang L., Berson S., Herzog S., Jamin S.; *RSVP Version 1 Functional Specifications*; Internet Draft 1996

[FJ94] Floyd S., Jacobson V.; *Synchronization of periodic routing messages*; IEEE/ACM Transactions on Networking, Vol.2, No.2, April, 1994

[MITZ94] Mitzel D., Estrin D., Shenker S., Zhang L.; *An architectural comparison of ST-II and RSVP*; Infocom'94, June, 1994

[PAN97] Pan P., Schulzrinne H.; *Staged Refresh Timers for RSVP*; IBM Research Report RC 20966 (09/02/1997), Global Internet 1997

[PAN98] Pan P., Schulzrinne H.; *YESSIR: A Simple Reservation Mechanism*; IBM Research Report RC 20967 (09/02/1997), Submitted for Publication, 1998

[SCHW97] Schwantag U.; *An Analysis of the Applicability of RSVP*; Diploma Thesis at the Advanced Network Technology Center, University of Oregon and Inst. of Telematics, University of Karlsruhe

[SHEN96] Shenker S., Wroclawski J.; *General Characterization Parameters for Integrated Service Network Elements*; Internet Draft 1996

[SHEN97] Shenker S., Partridge C., Guerin R.; *Specification of the Guaranteed Quality of Service*; Internet Draft 1997

[TOPO90] Topolcic C.; *Experimental Internet Stream Protocol: Version 2(ST-II)*; Internet RFC 1190, October 1990

[WROC96] Wroclawski J.; *The use of RSVP with IETF Integrated Services*; Internet Draft 1996

[ZAPP96] Zappala D.; *RSRR: A routing interface for RSVP*; Internet Draft 1996

[ZHANG93] Zhang L., Deering S., Shenker S., Zappala D.; *RSVP: A New Resource Reservation Protocol*; IEEE Network, September 1993

[ZHANG96] Braden R., Zhang L.; *RSVP Version 1 Message Processing Rules*; Internet Draft 1996