Scalable Resource Reservation Signaling in the Internet

Ping Pan

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2002

ABSTRACT

Scalable Resource Reservation Signaling
in the Internet

Ping Pan

Resource reservation protocols were originally designed to signal end hosts and network routers to provide quality of service to individual real-time flows. More recently, Internet Service Providers (ISPs) have been using the same signaling mechanisms to set up provider-level Virtual Private Networks (VPNs) in the form of MPLS Label Switched Path (LSP). It is likely that the need for reservation signaling protocols will increase, and these protocols will eventually become an indispensable part of Internet service. Therefore, reservation signaling must scale well with the rapidly growing size of the Internet.

Over the years, there have been debates over whether or not there is a need for resource reservation. Some people have been advocating over-provisioning as the means to solve link congestion and end-to-end delay problems. The over-provisioning argument is largely driven by the expectation that the bandwidth price will drop drastically. From our investigation, however, we found that many end users have not been benefiting from over-provisioning: the current Internet has bandwidth bottleneck links that can cause long-lasting congestion and delay. At the same time, leased line cost has not been reduced sufficiently in a timely manner for many network providers to deploy high-speed links everywhere in their networks.

Applying resource reservation brings many benefits to the network users. Unfortunately, the current resource reservation framework has scalability problems in terms of storage, bandwidth, message processing and manageability. To address these problems, we first evaluate methods that are designed to improve the scaling properties in RSVP. Though some of the methods can reduce protocol processing overhead substantially, they do not reduce the total number of reservations in the network. Thus, we argue that merely enhancing the existing signaling protocols may not be sufficient.

Generally, scalability problem can be solved by building a hierarchy. Resource reservation signaling is no exception. Depending on traffic behavior and service requirements, we propose a hierarchical reservation model that will support reservation signaling capability at end-user's application layer as well as at network provider's backbone level. In the model, end users may use lightweight signaling protocols to setup reservations for short-lived real-time applications. Within the network, service providers, based on bilateral agreements, establish long-lasting and more static reservation "trunks" among each other. At the network edge or border, end-user reservations are aggregated into provider's reservation "trunks", depending on user's qualification and network resource availability.

To explore our understanding on lightweight signaling, we introduce YESSIR, a simplified application-layer reservation protocol. It is designed to establish reservations for real-time streaming traffic. To simplify the processing at routers, YESSIR uses one-pass signaling sequence and allows data senders to initiate reservations. YESSIR also uses partial reservation and reservation retry techniques to speed up the setup. Our implementation results show that with proper protocol design and implementation, network routers can support a large number of user reservations (10,000 reservation requests per second on a FreeBSD prototype).

One of the most challenging aspects on provider-level signaling is that the protocol needs to be applicable and scalable to potentially all network providers in the Internet. After evaluating traffic traces from the Internet backbone, we derive a sink-tree algorithm, where the reservations from other providers following inter-domain routing path to a destination provider's network form a tree, rooted at the destination provider's border router. The sink-tree approach has the property that the maximum number of reservations at network routers is always $O(N)$, where N is the total number of routing domains in the network. This should reduce the total number of inter-domain reservations to a manageable level. We present an inter-domain reservation protocol, BGRP, that is based on the sink-tree algorithm. BGRP also has several built-in features that allow fast setup and make it resilient to route flapping.

Contents

List of Figures

List of Tables

Acknowledgments

The work presented in this dissertation was inspired and supported by my advisor, Professor Henning Schulzrinne. I am deeply grateful for his guidance, his patience, his understanding, his insight, and his friendship. The calibration of ideas, debates and frequent exchange of suggestions have been invaluable to me and enabled me to conduct sound and fun research. It has been a privilege to be advised by him.

I wish to thank the members of my dissertation committee, Professor Nick Maxemchuk, Professor Vishal Misra, Professor Dan Rubenstein and Professor Roch Guérin.

Many people have influenced me greatly since the beginning of my professional career in 1991. Tim Rolfes, Marc Cochran and Bill Guckle, who were my mentors, colleagues and friends during the years of building NSFnet routers in IBM, had taught me to roll-up the sleeves and become an engineer. Dr. Roch Guérin, my manager and mentor while I was in IBM T. J. Watson Research Center, has broadened my professional horizon greatly. I am forever indebted to Roch for showing me the discipline and the ethics in research. I owe many thanks to Dr. Ellen Hahne for her patience and technical advice while I was working at Bell Labs, Lucent.

I have enjoyed the privilege of sharing spirited discussions with many other researchers and developers over the years, in particular, Alex Birman (IBM Research), Robert Haas (IBM Research), Vinod Peris (IBM Research), Liang Li (IBM NHD), Bernard Suter (Bell Labs), Rohit Dube (Bell Labs), Luca Salgarelli (Bell Labs), Jim Murphy (Juniper Networks), Yakov Rekhter (Juniper Networks) and Der-Hwa Gan (Juniper Networks). I also thank many people in the Internet technical community, whose opinion, suggestions, requirements and disagreement have helped to shape the outline of this dissertation.

I gratefully acknowledge the support of IBM (Dr. Rick Bovie and Dr. Paul Stirpe) and Bell Labs (Dr. David Lee, Dr. Al Aho and Dr. Tom LaPorta) for allowing me to pursue my Ph.D. degree while working full-time. I thank my fiancée, Jenny Yeh, for her loving, kindness, understanding and all the “headless” muffins and extra-strong coffees during the crunch time. Thanks also go to many of my friends (including those who keep asking “when are you going to finish?” and “why?”) and my IRT Lab colleagues, in particular, Jonathan Rosenberg, Xin Wang and Maria Papadopouli, for encouraging me over the years.

Finally, I dedicate this dissertation to my father, Professor Cheng-Sheng Pan, and my mother, Dr. Chung-Xiou Wang, for giving up so much in China and coming to America years ago, so that I could have a better future. Despite hardships and illness over the years, they have been encouraging me and reminding me to follow my heart and go after the things that worth a while. I would not have completed this work if not for them. *I hope I have made them proud!*

Chapter 1

Introduction

The Internet was designed to provide connectivity to computers so that the end users can exchange data. It offers *best-effort* service, in which the network does not guarantee anything, not even the delivery of the data. Originally, the Internet was relatively small in terms of nodes, and users, and was managed by a small number of research institutes. The original user applications, such as, telnet, e-mail and FTP, do not have real-time delivery requirements, where user packets do not have to be delivered over the network within a delay bound. Consequently, the network was simple to design, and easy to deploy.

However, with time, the Internet has rapidly expanded in several dimensions:

More users: The Internet is expanding at a rate of 80-100% per year in terms of users and computers on line. Some [5] have estimated that there would be nearly 2.5 billion devices on the Internet by 2006 and by 2010 half the world's population will have access to the Internet. The rapid growth thus presents a *scalability* challenge in network design, deployment and operation.

More services: To continue generating revenue and reduce operational cost, media providers, traditional telephone companies and Internet service providers (ISP's) need to create and converge various services into the Internet. The new service features include support for voice, computer games, radio and, potentially, television. Subsequently, delivering *real-time* data will becomes a new requirement for the Internet. Note that radio and television are not necessarily delay-sensitive when using playback techniques, such as buffering.

Private network integration: To reduce the cost of network operation, many privately leased lines and private networks can be replaced with virtual private networks (VPNs) over the public Internet. VPN clients require secure and predictable data delivery. This adds another constraint to the existing Internet model.

Network heterogeneity: In addition to the legacy networks, such as TDM and ATM, in the next few years, the Internet will have ultra-fast optical links as well as very low speed wireless access connections. Compounded with the rapid network expansion measured in hop-counts and ISP-counts, user traffic is likely to traverse over more network links that can have different bandwidth spectrum, noise level and media characteristics. As a result, transmission latency and jitter may cause serious problems to some of the end-user applications.

ISP isolation: After NSFnet, the government-sponsored Internet backbone, was officially turned off in 1995, commercial ISP's began to proliferate and became increasingly competitive. Today, there are thousands of ISP's world-wide. They compete against each other by offering more services (such as IP telephony and virtual private networking) and better services (more predictable delay and more bandwidth) *within* the bounds of the ISP. Unfortunately, this practice has failed to take into consideration the end-to-end traffic flows that may travel through multiple ISP networks. Today's Internet consists of many well-engineering "islands" connected together by some weak links. We will provide evidence for ISP isolation in Chapter 2.

Consequently, as new services have been deployed throughout the Internet, we are likely to run into the following limitations:

- the network cannot efficiently provide end-to-end service guarantees to real-time inter-active traffic;
- there is no efficient and feasible control mechanism in place that can offer service guarantees beyond a single provider's network.

One way to overcome these limitations is to support *resource reservations*, in which network resources (*e.g.*, bandwidth, packet buffers) are set aside for a particular flow. Reservations can be user-to-user and network-to-network. To obtain a reservation, clients signal the network to request the amount and quality of service; the network then decides to whether or not it can satisfy this request. In this dissertation, we study several scalable and efficient approaches in reservation signaling.

This introductory chapter first explains some basic terminology and the network model that the work is based on. We then provide an overview of the rest of the dissertation.

1.1 Network Model and Terminology

The current Internet structure is based on a distributed architecture that is operated by many commercial providers, such as UUnet, Sprint and AT&T, interconnected at various traffic exchange points. Figure 1-1 illustrates the general structure of the Internet.

Each provider's network has a number of connection points called *POPs* (Points of Presence). Customers are connected to the providers via the POPs. Providers that have POPs in multiple regions are called *backbone providers*, as oppose to *regional providers* that only have POPs in a single region. A typical POP has a large number of modem ports for dial-up, Frame Relay, DSL or customer T1. For instance, in 2000, the UUnet network had more than 2,500 POPs throughout North American, Europe and the Pacific Rim, and supported more than 1.6 million modem ports [6].

Providers connect to other providers at one or multiple points. These interconnection points are called *NAP's* (Network Access Points), and enable users of one provider to reach users of another provider. A typical NAP is a high-speed network (such as FDDI or Gigbit-Ethernet), or an ATM switch to which a number of routers can be connected for the purpose of traffic exchange. In 2000, there were more than 50 public NAP's [7] in North America. Small NAP's, such as Mountain Area Exchange in Denver, Colorado, typically interconnect less than a dozen regional networks, whereas some of the larger NAP's (such as MAE East) provide connectivity to more than 100 providers.

Providers are also connecting each other through *private peering*, where two providers

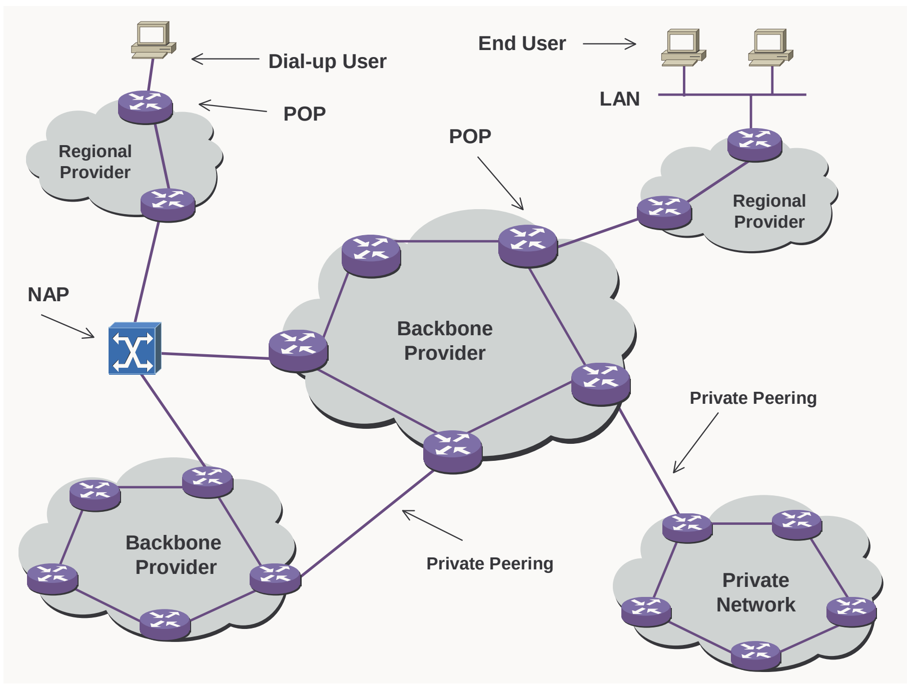


Figure 1-1: A generic Internet structure. Each network consists of many routers. For illustration purposes, we only show the edge/border routers that are functioning as POP's. The Internet is built on top of circuit networks, that switch traffic using Frame Relay, ATM or SONET. For clarity, we only display the network that is visible from IP routing protocol perspective.

are directly connected by high-speed links. The benefits for private peering are cost saving and eliminating inter-provider traffic latency. In recent years, providers have moved from inter-connected through NAP's to through peered links. It has been reported that some ISP's transmit about 70-80% of their traffic through private peering [8].

Private corporate networks access the Internet at private POP's, or through private peering to large ISP's. It's difficult to estimate the capacity and internal structure of private networks, however, it is believed [9] that most of the current corporate networks are structured either in mesh or star topology, and mostly running at T1 (1.54 Mb/s) or T3 (45 Mb/s) link speed.

Most of the today's Internet *end users* are residential computer users. Most of them access the Internet via low-speed modems, though this situation is likely to change as DSL and cable modem are being deployed. Potentially, another low-speed end-user population comes from cellular networks, where the connection speed is in the range of 10 kb/s in GSM. However, 3G technologies can improve wireless user data rate to as high as 2 Mb/s.

The Internet is connected by *routers*, a type of special-purpose computer, that can direct traffic from one place to another. The protocols that dynamically inform the routers of the paths that the data should take are called *routing protocols*. Routing protocols can be categorized into two types. The protocols that distribute routing information within a regional or backbone network are *IGP* (Interior Gateway Protocols). Some of the typical IGP are RIP [10], OSPF [11] and IS-IS [12]. Both OSPF and IS-IS are well designed, and have no stability and scalability problems in reasonably structured networks [13], while RIP has been phasing out in production networks. The protocols that exchange routing information between provider networks are *EGP* (Exterior Gateway Protocols). Currently, the de facto EGP is BGP4 [14] and several of its important extensions [15, 16, 17, 18, 19]. After years of experimenting and engineering, BGP is reasonably stable and scalable, although route instability [20] and slow route convergence [21] remain significant problems in the backbone networks today.

The entire Internet is divided into a number of *routing domains*, *domains*, or *Autonomous Systems* (AS's). All the routers in an AS share the same routing policy, and are under single technical administration. Each AS could be a collection of IGP's working together to provide interior routing. To the outside world, the whole AS is viewed as one single entity. Each AS has a unique identifying number throughout the Internet, which is used by EGP during route distribution. In most cases, each provider network has one AS, however, some of the larger backbone providers can have multiple somewhat independent networks, and use several AS's in routing.

1.2 Dissertation Overview

Resource reservation has been a very controversial issue among academic scholars and network operators. The next chapter examines the question of whether or not the Internet is better off with resource reservation from several aspects. An analytical model shows that reservation-capable networks provide better performance for real-time traffic than best-effort-only networks. A study on today's Internet traffic volume, bandwidth pricing, and network topology indicates that network heterogeneity could be a significant problem that prevents ISP's from delivering predictable services to end users. We also evaluate some of the potential network applications and conclude that supporting resource reservation is a key to deploy those applications.

RSVP [22, 23] is the most recent resource reservation protocol for the Internet, however, it has many scaling problems. To improve scalability, Chapter 3 describes a mechanism called *staged refresh timer* that can reduce the total number of RSVP messages in the network, while improving hop-by-hop message delivery. The enhancement is backwards-compatible and can be easily added to current implementations. We also evaluate several other proposals that are aimed to enhance RSVP scalability.

Running a single protocol to setup all the reservations in the Internet is bound to have scalability problems, as the Internet size is growing exponentially at a rate that is similar to the one that has produced "Moore's Law" in semiconductors. Chapter 4 presents a hierarchical architecture for providing resource reservations in the Internet. At the bottom level of the hierarchy, end-user applications can initiate, modify and terminate reservations. The key requirements for application-layer reservation protocols are that reservations must be simple to process, and fast to set up. At the top of the hierarchy, network providers can create large and more static reservation *trunks* between each other. Inter-provider reservations are created based on bilateral or multilateral agreements. The key requirements here are: the total number of reservations must be manageable; the reservations have to be robust during frequent routing changes.

Chapter 5 investigates two key issues in designing reservation protocols: signaling sequence (*i.e.*sender-initiated vs. receiver-initiated), and partial reservation. Both contribute directly to protocol's scalability. In particular, we investigate techniques that allow routers to quickly recover from partial reservations.

In Chapter 6, we present the design, implementation and performance of a lightweight signaling protocol, YESSIR (YEt another Sender Session Internet Reservations). YESSIR is designed to setup resource reservations for real-time streaming applications that use

RTP [24]. Senders generate requests to reduce processing overhead, and it uses user-to-user *soft state* to maintain reservation states. It also supports the features such as sharing reserved resource among a group of senders, and network resource advertisement. More importantly, it extends the all-or-nothing reservation model to support partial reservations that improve over the duration of the session.

Chapter 7 presents a provider-level reservation protocol called BGRP (Border Gateway Reservation Protocol). It manages network resources at routing domain (AS) level, and interacts closely with BGP. BGRP can aggregate all the traffic going to the same destination AS. Therefore, it keeps the number of the reservations in backbone routers to $O(N)$, where N is the total number of AS's. Since BGRP is built on top of BGP, it is not effected by route looping, flapping and convergence problems.

We conclude in Chapter 8.

Chapter 2

Do We Need Resource Reservation?

2.1 Background

In 1993, Zhang *et al.* [22] described *resource reservation* as an important component for providing service guarantees in the Internet. RSVP [23] has been proposed as the signaling protocol that is responsible for setting up reservations for user packet streams in the network. However, over the years, many people [25, 26] have been debating over whether or not there is a need for resource reservation in the Internet in the first place. The arguments against resource reservations include:

Over-provisioning is good enough: Because bandwidth is becoming a cheap commodity, the networks can afford to be over-provisioned to eliminate link congestion and to reduce packet transmission latency. This popular argument seems to be confirmed by the fact that several large backbone providers, UUnet and Verio, have been deploying high-speed links (up to OC-192, or 10Gb/s, in 2000), and used over-provisioning to overcome link jitters and congestion. It is important to note here that this argument is based on the expectation that, as more advanced technology (such as optical networking and gigabit Ethernet) becomes available, the bandwidth price will be driven down rapidly and consistently in the near future.

Applications can adapt: End users can use various adaptation mechanism, such as buffering, signal reconstruction, and adaptive compression, at the application layer to adapt to network delay and packet losses. For example, NeVot [27] and rat [28] have a built-in buffering mechanism to adjust play-out delay, and vat [29] and rat can reconstruct lost signal.

Real-time traffic volume is too small: Most of the network traffic is *elastic*, and has loose time constraints. Most of the multimedia data (video and audio) transmission should not be considered as real-time either, since users have the option to download or cache data in advance. Therefore, engineering a network to accommodate small amount of inter-active real-time traffic does not seem to be economical.

Adds QoS support processing cost to routers: Processing control messages, maintaining reservations states and running admission control contribute to heavy processing and memory overhead at routers. This argument has gained a lot of support among engineers during the development of RSVP and IntServ [30, 31, 32, 33].

Adds complexity to service providers: Reservations categorize user traffic into multiple “flows”, each may associate with a different accounting, authentication and billing procedure, and needs to be handled separately by the ISP. Given that the number of “flows” can be very large, a scalable management and billing system can be complicated and costly to develop and deploy.

Do we need to use reservations in the Internet? The answer to this question depends on many factors, such as, bandwidth cost, Internet growth rate, service deployment process, and the emergence of “killer” applications in the future that can significantly change traffic pattern and traffic dynamic. In this chapter, we argue the necessity of resource reservation from several aspects, and evaluate some of the flaws in the above argument.

2.2 Performance Evaluation for Real-Time Applications

Shenker and Breslau studied [34, 35] the performance of real-time applications in the Internet through the use of a utility function formulation and several simple models. They concluded that depending on traffic pattern, reservation-capable networks retain significant advantages over best-effort-only networks, no matter how cheap bandwidth is.

Two types of real-time application are defined here: *rigid* and *adaptive*. The rigid applications are extremely sensitive to delay and have hard real-time requirements. These applications need their data to arrive within a given delay bound. An example of such applications is the traditional phone service. Adaptive applications, on the other hand, can be audio and video applications that are implemented to tolerate occasional delay-bound violations and dropped packets. A typical adaptive application is VoIP, that can have adjustable delays.

The utility functions for both rigid and adaptive applications are illustrated in Figure 2-1. The utility function for a rigid application flow is

$$\pi(b) = \begin{cases} 0 & \text{for } b < \bar{\mathrm{b}} \\ 1 & \text{for } b \geq \bar{\mathrm{b}} \end{cases}$$

where $\bar{\mathrm{b}}$ is the bandwidth needed for real-time applications.

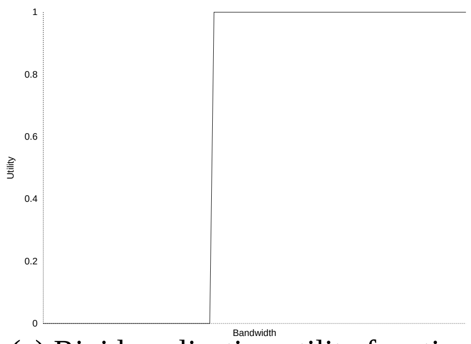
For applications that can adapt to both changes in rate and delay, the transition in utility will be smooth from low bandwidth to high bandwidth. For example, the utility function can be characterized as the following:

$$\pi(b) = 1 - e^{\frac{-b^2}{\kappa + b}}$$

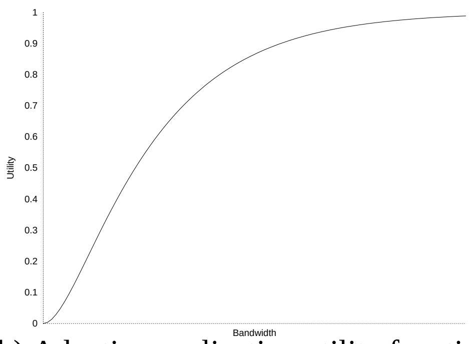
where κ is a constant. Similar to other elastic applications (file transfer and email), at high bandwidth, there is only a marginal performance gain with additional bandwidth. Nevertheless, the adaptive applications always have an intrinsic bandwidth requirement, and this intrinsic generation rate is independent of the network congestion. The performance degrades badly as soon as the bandwidth share becomes smaller than the intrinsic rate. Thus the network can become overloaded, though the applications can adapt to network congestion.

The fundamental difference between a best-effort-only network and a reservation-capable one is that, in the former, user flows are never denied access to the network, whereas a reservation-capable network can deny reservation requests.

Let $P(k)$ be the possibility that there are k flows requesting service, and C be the link



(a) Rigid application utility function



(b) Adaptive application utility function

Figure 2-1: The utility function for rigid and adaptive applications.

capacity. The average number of the requesting flows in the network is $\bar{k} = \sum_{k=0}^{\infty} P(k)k$.

Hence, the total utility of a best-effort-only network is given by:

$$V_B(C) = \sum_{k=1}^{\infty} P(k)k\pi(\frac{C}{k})$$

For a reservation-capable network that can admit at most $k_{max}(C)$ flows, the total utility becomes

$$V_R(C) = \sum_{k=1}^{k_{max}(C)} P(k)k\pi(\frac{C}{k}) + \sum_{k=k_{max}(C)+1}^{\infty} P(k)k_{max}(C)\pi(\frac{C}{k_{max}(C)})$$

The normalized utility functions for best-effort-only and reservation-capable networks are respectively,

$$B(C) = \frac{V_B(C)}{\bar{k}}$$

and

$$R(C) = \frac{V_R(C)}{\bar{k}}$$

By modeling $P(k)$ as Poisson, exponential and algebraic load distributions[1], they observed the following important insights:

- **Performance and bandwidth benefits:** The best-effort-only networks may be simple to build, but they require additional bandwidth in order to match the performance of the reservation-capable networks. This additional bandwidth is defined as the *bandwidth gap*, $\Delta(C)$, such that, $R(C) = B(C + \Delta(C))$.

  When evaluating the utility functions and the bandwidth gaps for both rigid and adaptive applications, it was found that, not surprisingly, rigid applications perform much better in a reservation-capable network than that in a best-effort-only network. However, surprisingly, for Poisson and exponential load distribution, the performance results for adaptive applications are almost identical in both best-effort-only and reservation-capable networks. The bandwidth gap disappears at high link capacity region. This behavior demonstrates the advantage that adaptive applications can tolerate network overload conditions, and raised the issue whether reservations have any advantage. However, when using algebraic load distribution in the model, the bandwidth gap between two networks is a constant[2] regardless of link capacity. This implies that the best-effort-only networks always require more bandwidth than reservation networks to match their performance.

- **Cost and bandwidth trade-off:** The reservation-capable networks impose the burden of additional complexity compared to best-effort-only networks. If the cost of extra complexity in building a reservation-capable network is converted to the cost of additional bandwidth, we can have the following welfare function:

[1]In the paper [35], the Poisson distribution is defined as $P(k) = \frac{\nu^k e^{-\nu}}{k!}$, and the exponential distribution is $P(k) = (1 - e^{-\beta})e^{\beta k}$. The algebraic distribution is defined as $P(k) = \frac{\nu}{\lambda + k^z}$. Note, when setting λ to be zero, it becomes a simple Pareto distribution, which has been used to model the *self-similarity* behavior of Internet traffic. For detailed mathematical analysis, refer to the paper directly. We only present the model and the results here for clarity.

[2]For $z \to 2^+$, the constant is e.

$$W(p) = V(C(p)) - p\,C(p)$$

where p is the cost per unit bandwidth. When setting the welfare of reservation-capable and best-effort-only networks to be equal, (*i.e.*, $W_R(\bar{p}) = W_B(p)$), $\bar{p}$ is the cost per unit bandwidth in reservation-capable networks, and the ratio $\gamma(p) = \frac{\bar{p}}{p}$ thus indicates the added complexity in reservation-capable networks relative to the best-effort-only networks.

It was shown that when using adaptive applications, for Poisson and exponential distributions, $\gamma(p) \to 1$ as $p \to 0$. This implies that as bandwidth becomes cheaper, $\bar{p} \to p$, *i.e.*, if adding reservation capability to the network introduces any additional per-unit bandwidth cost, it would make the best-effort-only network the preferable choice.

However, in contrast, in the algebraic case, $\gamma(p)$ is always greater than one, and does not converge. This implies that the best-effort networks cannot perform as well as the reservation-capable networks, no matter how inexpensive the bandwidth becomes. Thus, when by introducing a small amount of additional per-unit bandwidth cost, a reservation network is always the preferable choice over a best-effort-only one.

In conclusion, it has been shown that there is a significant performance gain between best-effort-only and reservation-capable networks for rigid applications. For adaptive applications, with Poisson and exponential load distributions, there is no strong need for reservations, but with algebraic load distribution, reservations yielded significant benefits. Therefore, the need for reservation-capable networks depends on the traffic pattern in the future.

2.3 Traffic Condition in Today's Internet

Despite the claims by some providers [36, 37] that their networks have little jitter and congestion, end-user traffic suffers from significant delay and jitters. From the Internet traffic dynamic studies conducted by Vern Paxson [38, 39], it was shown that by sampling over 20,000 TCP traces from 1994 to 1997 among 35 widely spread sites:

- Overall packet loss was between 2.7% to 5.2% randomly during the measurement period.
- *Available bandwidth* is defined as the proportion of the total network resources that were available by an end-to-end connection itself. A value of 1 means that bandwidth is available on all physical links that are used by the connection, whereas a value of 0 indicates no link bandwidth. The was shown that the available bandwidth varied widely, from very little to almost 1.
- The end-to-end packet delay varied from 100 to 1,000 ms, but extended out quite frequently to much larger times.

The results indicate the following: the Internet has both high-bandwidth links, where packets can sail through without any disturbance, and *bandwidth bottleneck links*. When packets go through the bottleneck links, the transmission can take a very long time.

The range of the delay variation was particularly troublesome: to transmit one-way voice data, the maximum allowed end-to-end delay is approximately 200 ms [40]. Let's estimate that the total time for packetization and coding/decoding [41] is approximately 100 ms. This leaves approximately 100 ms, which is the lower bound in Paxson's measured delay variation, as the total network latency allowed for voice packets. This implies that, at worst, the end-user applications may have to absorb as much as 900 ms of delay. This kind of delay variance is not acceptable for real-time voice traffic.

Hence, it is reasonable to claim that, in order to support delay-sensitive end-user applications in the Internet, unless the network can *always* transmit user packets over the links that have *plenty* of bandwidth, the applications will suffer from jitter, or excessive delay.

This raises the following issues: Where are the bandwidth bottleneck links? Can we just over-reserve the bottleneck links? How real is the problem that end-user traffic actually travels through the bottleneck links? We will try to provide some insights to these questions in the following sections.

2.3.1 Bandwidth Bottlenecks

Network links cover a large spectrum of speeds. Typically, backbone networks have link speeds ranging from OC-3 (155 Mb/s) to as high as OC-192 (10 Gb/s), and are likely to operate at higher speed in the future, as the next-generation optical equipment is being deployed.

Within private and local-area networks, each user normally has at least 10 Mb/s (*i.e.*, switched Ethernet LAN's) of bandwidth to share, normally with a small number of other users. Most of the residential dial-up users run at 56 kb/s modem speed. With the emergence of high-speed LAN's (100 Mb/s Ethernet) and DSL/cable modem technologies, end users will eventually have more bandwidth to access the Internet. For wireless users, cellular (GSM) technology can only provide approximately 10 kb/s of bandwidth. However, in 3G, user data rates vary extensively depending on the technology (GPRS, 3G-1x, HDR, etc.). Wireless data rates can range from 64 kb/s all the way up to almost 2 Mb/s.

From recent measurement results [4], it was estimated that private, local-area and backbone networks are lightly loaded, as shown in Table 2.1. The particular reason for light traffic load within corporate networks is to maintain a low transaction latency within the network. This suggests that *as long as user traffic stays in the same network, latency and jitter are not a problem.* Hence, if there is any congestion in the Internet at all, it must come from places that interconnect subnetworks.

Network Type	Average Utilization
Backbones	10-15%
Private Networks	3-5%
LAN's	1%

Table 2.1: Network utilization estimation from [4].

We refer to the links that connect private networks to the Internet, or interconnect provider networks at POPs (Points of Presence), NAP's (Network Access Points) and private peering points, as *access links*. It is believed that the access links are the main points of congestion. The NAP's are particularly problematic. Unfortunately, most of the commercial ISP's do not publicly reveal the traffic statistics of their access links. We have to rely on traffic traces from several academic and research networks to piece together the link utilization conditions.

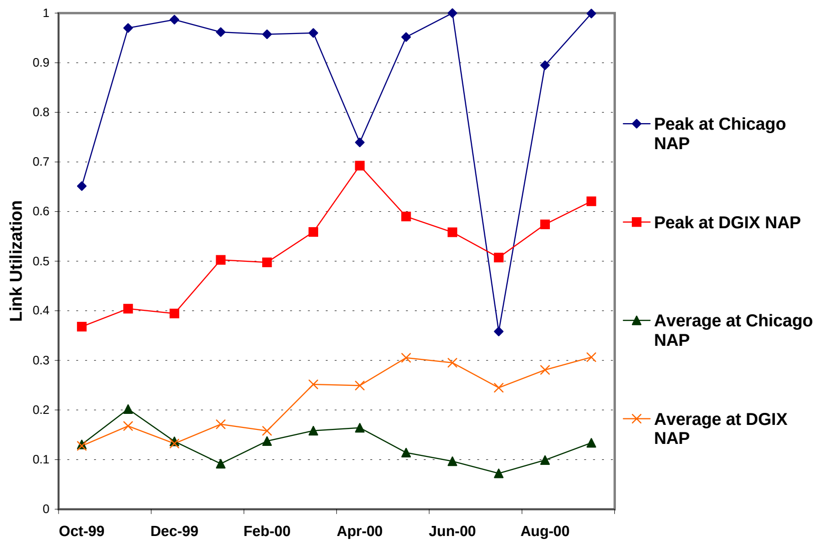
SWITCH [42] is a regional ISP that provides Internet connectivity to Swiss academic and research institutions. In 2000, the backbone consists of E3 (34 Mb/s) and OC-3 trunks. Table 2.2 shows the traffic statistics on some its access links. The measurement was made by collecting traffic counters from the routers every 5 minutes using SNMP. From the collected data, we notice that although the average link utilization is low, the peak utilization at the NAP and the direct peering links are quite high. In fact, the peak utilization at the CERN NAP had reached the maximum link capacity, since the NAP is operating on an Ethernet. Unfortunately, there is no data available for us to determine congestion duration.

Link Type	Average Utilization		Peak Utilization	
	In	out	In	out
Transatlantic OC-3 Link (Zurich - New York)	16.1%	5.6%	49.4%	32.1%
Trans-Europe OC-3 Link	10.5%	17.0%	23.9%	48.9%
CERN NAP, Geneva (100M Ethernet)	13.7%	25.1%	70.3%	89.6%
Direct Peering (to Swisscom) (100M Ethernet)	3.0%	12.5%	64.8%	67.0%

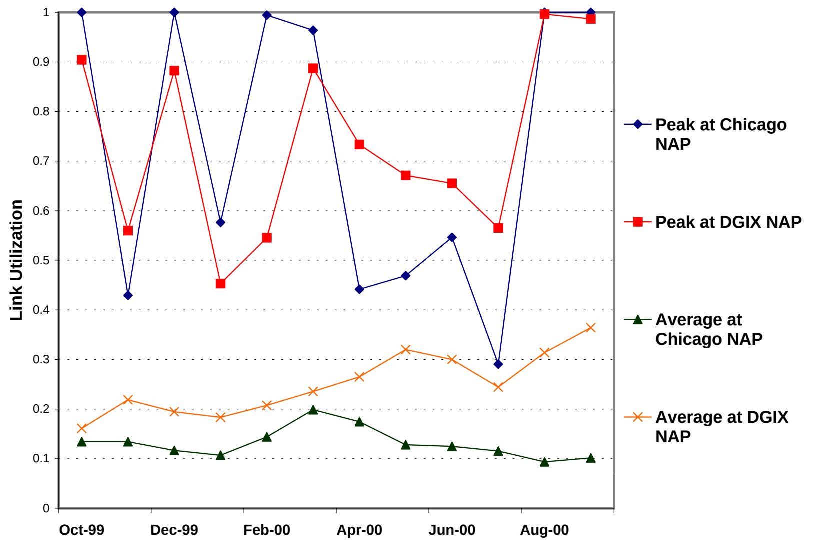
Table 2.2: Monthly link utilization in SWITCH access links on September, 2000.

NORDUnet [43] is a reasonably sized backbone network that connects Nordic countries to the rest of the world. The backbone itself is a mix of OC-3 and OC-12 trunks. NORDUnet connects to several NAP's, including the Chicago NAP, and is peering with several other large provider backbones, such as TeleGlobe, Telia and FUNET. Most of the access links are OC-3 trunks, except that the Chicago NAP link runs at DS-3 speed, and the link to DGIX NAP is a 100 Mb/s Ethernet connection. Similar to other providers, NORDUnet uses multiple links to peer with other providers, and traffic to and from other providers is evenly distributed across all the links. We selected the 12 busiest links out of a total of 36 access links in the backbone, and analyzed the peak and average link utilization shown in Figures 2-2, 2-3 and 2-4.

We observe the following interesting results:

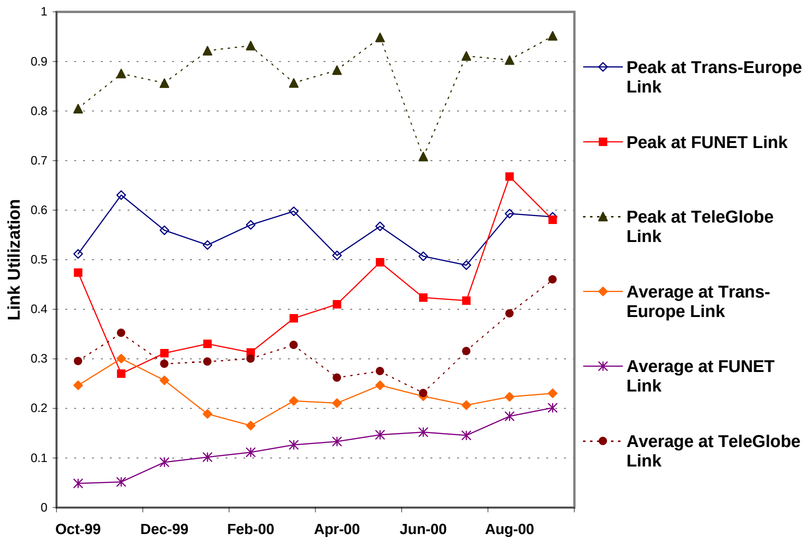


(a) Input traffic link utilization at the NAP’s.

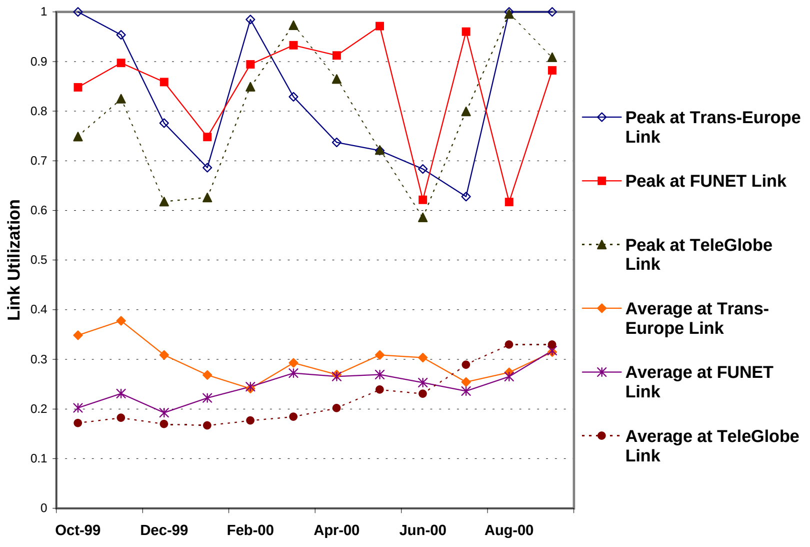


(b) Output traffic link utilization at the NAP’s.

Figure 2-2: NORDUnet traffic average and peak link utilization at the NAP’s from October 1999 to September 2000. Note that the DGIX NAP runs on a 100 Mb/s Ethernet, that has a maximum link utilization of 70-80%.



(a) Input traffic link utilization with peers.



(b) Output traffic link utilization with peers.

Figure 2-3: NORDUnet traffic average and peak link utilization with three different peers from October 1999 to September 2000. NORDUnet has multiple peering links to FUNET and TeleGlobe. The data here is the average from those links.

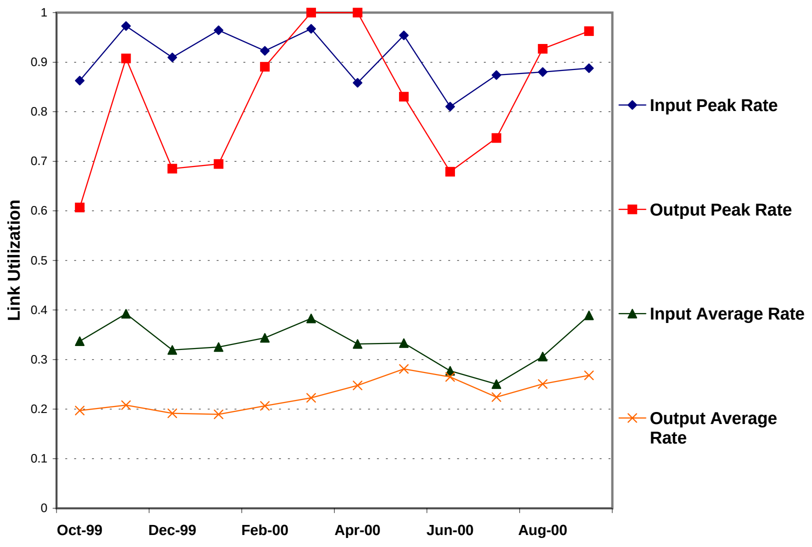


Figure 2-4: NORDUnet traffic average and peak link utilization on the transatlantic links from October 1999 to September 2000. NORDUnet has three OC-3 transatlantic links. The data here is the average of those links.

- Both NAP (Figure 2-2) and private peering links (Figure 2-3) can get congested. This is somewhat surprising, since it has been perceived that unlike NAP's, private peering links are better managed and thus have less congestion.

- Transatlantic links are a part of the provider's backbone, and have very high peak utilization (Figure 2-4). This implies that there are bottleneck links within provider networks, and user traffic that stays within the same provider's network can also get congested.

- Most importantly, the average link utilization is quite low, around 20% to 30%, however, the peak link utilization is high on all links. Some of the links had reached 100% link utilization. The congestion occurrence varies depending on links. From the statistics maintained at NORDUnet, the Chicago NAP link operated at peak only once in August, 2000, which lasted approximately 7 hours. The DGIX NAP link

> operated at peak (over 80% link utilization) five times in September, 2000, and only once in August, 2000. Each congestion in DGIX lasted between around 1 hour to as many as 10 hours. We have also observed frequent and long-lasting congestion on several peering links. During August and September, 2000, the TeleGlobe links had operated at peak (over 80% link utilization) almost every workday. Each peak lasted about 8 hours.

To understand the reasons for link congestion, we extensively studied the link utilization history[3]. We discovered that network links are being frequently re-configured and upgraded. One of its reasons is that providers regularly re-arrange or shift user traffic between links to balance traffic load and optimize bandwidth usage. For example, the reason for the sharp traffic increase and subsequent link congestion on the TeleGlobe links (Figure 2-3-(a)) was that to reduce the traffic "flooding" from external users and to make bandwidth available for internal users, NORDUnet had shut off one of the three OC-3 links to the TeleGlobe backbone.

Since we do not have the access to many other ISP's traffic statistics, we cannot make the claim that the traffic conditions in NORDUnet network are typical for the rest of the Internet. However, from the statistics collected from several other ISP's, such as SWITCH [42], Above.net [44] and BBC [45], we discovered a very similar traffic behavior: *the average link utilization is always reasonably low, and many links are lightly loaded at all time. However, every network always has busy links (particularly, access links at NAP's and peering points) that have long lasting high bandwidth utilization.*

2.3.2 Getting Connected Is Costly

One obvious solution to reduce or eliminate the congestion and jitters problems is to add more bandwidth at congested links, such as the access links. Unfortunately, this is unlikely to happen any time soon for many networks, especially those operated by the smaller ISP's. This is because most of these links are leased from telephone companies, and are *very* expense.

[3]We acknowledge the kind and detailed response from Havard Eidnes at NORDUnet.

Although many people have predicted that bandwidth would become ”dirt” cheap, the reality is that leased line prices have not decreased *consistently* and *rapidly*.

Andrew Odlyzko in [46] had estimated the tariffed price for a T1 link from New York City to Pittsburgh, a distance of about 300 miles, which is about the average distance for long distance private line links. In 1987, the link was priced at $10,000 per month. After a consistent decrease for 5 years, the price had shrunk to $4,000 in 1992. However, from 1992 to 1998, the link price has climbed by over 50%, to $6,000 per month. What is surprising here is that the price increase has taken place during the same period at the peak of Internet commercialization, which has been one of the largest network infrastructure expansions in history: thousands of ISP’s were founded, and the demands of leased lines were skyrocketing.

In spite the bandwidth price increases in the 1990’s, compared with Europe, North America has much lower leased line prices [47]. At the end of 1998, for comparable distances (300 km) and bandwidth (2 Mb/s), US leased lines are four times less expensive than ones within the same European country, and sixteen times less expensive than international links within Europe.

An important factor that is contributing to high bandwidth price is the costs of cable installation. Normally, cable installation expense includes engineering and construction expense (such as digging), negotiating right-of-way (ROW) costs and fiber costs. It was estimated that for phone or cable companies to install cables, the average cost for installation is between $7 and $15 per meter [48]. However, to put the fibers in use (such as generating signal, building regeneration sites), the total cost becomes $50 to $70 per meter [49].

Not only are leased lines expensive, interconnecting ISP’s can be costly as well. Table 2.3 shows the connection fees at some of the NAP’s.

A network provider needs to pay neighboring providers very high prices for carrying its traffic. Recently, some North American providers [8] have claimed that the peering cost for a transit DS-3 link was $50,000 per month, and a OC-3 link costs up to $150,000 per month. To reduce costs, providers have begun to adapt the method of establishing *private peering* arrangements among each other, and have reduced the amount of traffic going through public NAPs. Typically, in a private peering arrangement, two connected

NAP	E-3 (34 Mbps)	DS-3 (45 Mb/s)	OC-3 (155 Mb/s)	OC-12 (600 Mb/s)
Chicago (Ameritech)	-	3,900	4,700	-
Bay Area (PacBell)	-	3,750	5,000	13,500
France (Parix)	4,086	-	4,458	-

Table 2.3: Monthly Cost (in US Dollars) to connect to a NAP in a 3-year term. The NAP's listed here use ATM PVC to guarantee bandwidth for peer-to-peer traffic. The NAP's that operate with Ethernet and provide shared bandwidth among peers have a much lower price.

providers share the cost of a single link, assuming that the traffic volume on the link are roughly equal.

For example, Telia, a backbone provider in Sweden, had analyzed their transit costs and recognized that approximately 85% of their traffic at MAE-East, a large public NAP, was to their transit providers (*i.e.*, the providers that are paid to carry Telia's traffic) and the remaining 15% was through peering relationships. By focusing on establishing peering relationships with the top 25 destination AS's, they shifted the mix to 70% through private peering, with the remaining 30% of traffic going through the NAP. The result was increased traffic efficiency and a reduction in the cost of transit.

In conclusion, we believe that, similar to many other commodity products, such as computer software and microprocessors, the rapid technological progress and the rising consumer demand *do not* necessarily translate to rapid price decrease in the market. Due the monopoly by a small number of major players (phone companies, cable companies, and large Internet providers) [50], and strict and protective government regulations [51], the price for bandwidth will not decrease fast enough.

2.3.3 The Vast and Flat Internet

So far, we have shown the existence of bandwidth bottleneck links in the network, many of which are located at network access points. But how much end-user traffic is actually going over these links?

The answer depends on the size and the geographic location of the provider networks. For example, UUnet has a very dense network infrastructure in the United States. When two UUnet users communicate with each other in US, it is most likely that their traffic stays within the same network, and won't go over bandwidth bottleneck links. However, for European, Asian and Australian users, a large percentage of their Internet traffic is actually with US networks. According to an estimate [52], 60% of Telstra (the dominant Australian provider) Internet traffic had been with US provider networks. Therefore, it's reasonable to assume that a large portion of end-to-end traffic traverse over multiple provider networks.

The average end-to-end router hop-count was 16 in a 1996 measurement [53]. As measured by Paxson [54], the longest end-to-end hop-count in the Internet was greater than 30 in 1997. However, looking at the router hop-counts alone does not reveal traffic conditions inside the network. In fact, it is when user packets traverse multiple providers, they are more likely to experience congestion at inter-provider access links.

A typical example is the following: suppose that users from regional network $\mathcal{A}$ need to communicate with users in regional network $\mathcal{D}$. Backbone provider $\mathcal{B}$ provides transit service for $\mathcal{A}$; backbone $\mathcal{C}$ for $\mathcal{D}$. Backbone $\mathcal{B}$ and $\mathcal{C}$ exchange traffic at a NAP or through private peering. Thus, end-to-end traffic involves all four $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$ networks, and has a *provider-hop-count* of 4. Of course, it is also possible that there are more than two transit networks ($\mathcal{B}$ and $\mathcal{C}$, as illustrated here) to interconnect sparsely located user populations.

Figure 2-5 shows the AS path information gathered at BGP border routers at the University of Oregon [55] and BELNET [56] [4]. The x-axis is the provider-hop-count (represented as AS length), and the y-axis is the percentage of the destination networks. The figure shows the number of providers that a user at the measured network has to travel in order to reach a destination network. Most of the destination networks are 3 to 5 providers away. This result has been further confirmed by data gathered by Telstra [2].

Here, we need to take into account the following two considerations: First, the AS information is announced by the destination networks, and distributed by BGP. Border routers can receive the same AS reachability data from multiple neighboring networks. It's up to the border routers to determine the most suitable routing path. Hence, the collected AS path information may not reflect the actual data forwarding path.

[4]The data was collected by Olivier Bonaventure of BELNET in December 1999. We acknowledge his timely response to our inquiry.

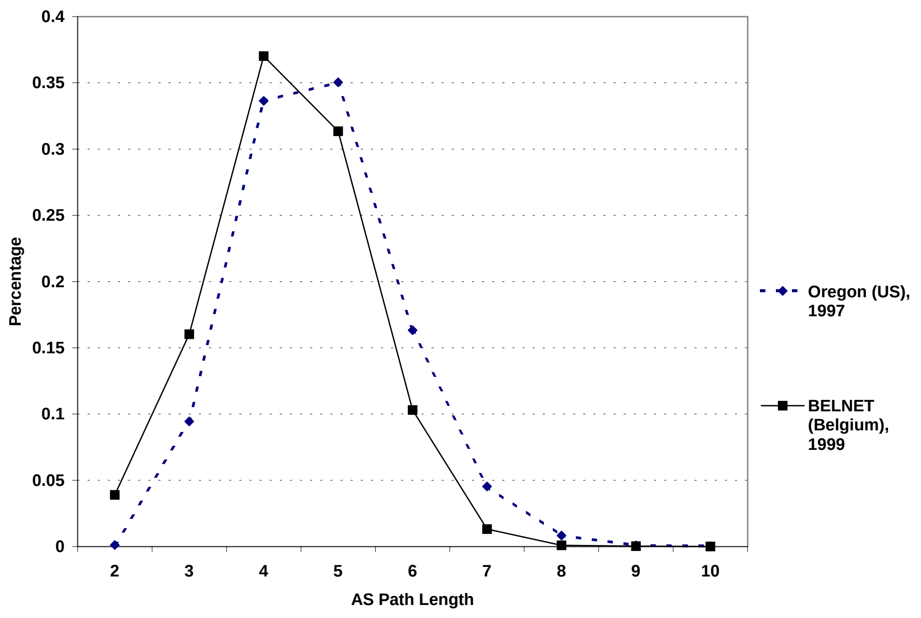


Figure 2-5: The AS path length information gathered from BGP AS-Path attribute.

The BGP routing information (such as route prefix and AS path) is aggregated at border routers. The reason for BGP aggregation is to reduce the total number of states maintained by the routers, and to localize the route flapping effects to a smaller portion of the network. It is possible that the measured AS path length has been aggregated and suppressed, and the actual AS length (or provider hop-count) may be longer.

The more network providers that user packets travel over, the more likely that they will go over bandwidth bottleneck links. Given the on-going rapid Internet deployment in the Far East and Latin America, and continuing growth in US and Europe, we predict provider hop-counts between large user populations will increase in the near future.

2.4 Sustaining Internet Traffic Growth

The Internet is in a transitional period. According to an 1998 estimate by Coffman and Odlyzko [9], Internet traffic has a 100% annual growth rate. We predict that, in the near future, some of the new applications may not only accelerate traffic growth, but change traffic characteristics as well. This will add more network deployment and management challenges to providers. Here are some of the emerging applications.

VPN traffic: In studies conducted by Coffman and Odlyzko [9, 52], it was shown that in 1997, the traffic carried by private lines and the Internet was 3,000 - 5,000 TB/month, and 2,500 - 4,000 TB/month, respectively. In 1999, there was 5,000 - 8,500 TB/month for private lines and 10,000 - 16,000 TB/month for the Internet. In 1997, the revenue for retailed leased lines for private non-Internet carriers companies was 10 billion dollars in US. The numbers imply that private networking is a profitable and growing business.

For this reason, nearly all network providers have entered the private networking business by offering VPN (Virtual Private Network) services over their Internet backbones. Currently, there are two trends in provider-based VPN: one is to interconnect existing private networks, such as Frame Relay and ATM networks, over the public Internet infrastructure, and the other is to replace some of the legacy private networks with the new IP-centric VPN technologies. In both cases, it requires the Internet to be able to duplicate the same secure and predictable services as in the private networks. Specifically, legacy services such as SNA require timely packet delivery and are extremely sensitive to packet loss. There is also a need for having a scalable and robust signaling mechanism that can establish and maintain those VPN connections.

Real-time data: Though streaming data with real-time requirements represents a very small portion of the total Internet traffic today, this situation can be quickly altered if Internet is to carry telephony voice traffic. In 1999 [52], the total voice traffic in US telephone systems was 40,000 TB/month, which is about four times more than the Internet traffic during the same period. This implies that, even with a small portion of the voice traffic going through the Internet, it can dramatically change traffic pattern and traffic dynamics in the network.

Some have argued that there is no good economical reason for introducing voice service to the Internet, because the net cost of using the Internet and the existing phone networks is pretty much the same. However, this is not a proper measure to evaluate the emergence of IP telephony technology. The most significant advantage that IP telephony can offer is *not* as a telephone service replacement, rather it is meant for service integration, where voice, messaging and audio/video streaming applications can be accessed by end users from a single networking interface. It thus forces the networks to add the capability to deliver user packets that have different criteria in terms of delay and throughput. Once again, we notice that it requires scalable and efficient signaling mechanisms to ensure appropriate services for the voice sessions within the Internet.

"Killer" applications: Network engineering and deployment take time. The past history tells us that user traffic triggered by some "killer" applications could potentially "flood" the network before the providers can respond and adjust their network capacity. For instance, due to Web browsing, the Internet traffic has an abnormal growth [52] that doubles every three or four months in 1995 and 1996, which represents a 100-fold traffic explosion over a two-year span. Another traffic explosion took place more recently. In 1999, some networks had observed abnormal traffic growth [57] due to Napster, a service that allows Internet users to exchange music files.

2.5 Scalability in Resource Reservation

It was shown previously that reservation-capable networks deliver better performance than best-effort-only networks for both rigid and adaptive traffic depending on the traffic assumption. However, reservation-capable networks require the routers to have admission control capability, which is believed to impose various scalability problems.

The scalability problem can be categorized into two aspects: *data-plane* and *signaling* (or *controlr-plane*).

In the data-plane, in a network that has many user flows, the processing overhead associated with real-time scheduling and queuing becomes non-negligible [58]. The overhead is caused by maintaining queues for each “micro” (per-user) flow and assigning packets to each queue. To reduce per-flow queuing overhead, several alternative architecture have been proposed, including the IETF DiffServ model [59] and the *dynamic packet state* [60] architecture. In the DiffServ model, routers simply implement a set of buffer management and priority-like queuing disciplines for each of a very small number of traffic “classes”, providing them with coarse grained rate guarantees. At the network edge, user flows are aggregated into these classes. It has been shown [61] that routers can indeed provide reasonably accurate rate guarantees and fair distribution of excess resources, with minimal impact on raw forwarding performance.

In this dissertation, we focus on the scalability issues related to reservation signaling. The scaling issues include the following:

- Memory utilization in routers (Section 2.5.1),
- bandwidth utilization for control messages (Section 2.5.2),
- control message processing cost, and
- manageability by network operators.

2.5.1 Router Memory Usage

When RSVP was first proposed in 1993, routers were not equipped with much memory due to the high cost of fast memory at the time. Router’s software was carefully designed and developed to make a good use of the limited memory to support rapid growing IP routes. For example, in 1993, a typical IBM NSFnet backbone router was configured to operate with 16 MB of 80 ns DRAM. During the implementation of RSVP, some developers realized that storing a single reservation could take as much as 500 bytes [62, 63]. Thus, to support 10,000 RSVP sessions would take at least 5 MB of memory. At the time, there were concerns among developers over RSVP’s large memory requirement.

However, in the intervening years, the cost of memory has gone down dramatically. Today’s routers are equipped with more memory: Cisco 12416 routers [64] can support up to 256 MB DRAM, and Juniper M-160 routers [64] are equipped with 768 MB of DRAM by default. Nowadays, memory usage is no longer a bottleneck for signaling protocols.

2.5.2 Bandwidth Utilization

Frequent control message exchanges between routers consume significant amounts of link bandwidth. This is particularly significant in soft-state based protocols such as RSVP. In RSVP, control message bandwidth B_{ctrl} is proportional to the total number of reservations, N, reservation message size S, and soft-state refresh period R:

$$B_{ctrl} = \frac{N \cdot S}{R}$$

Assume each reservation is associated with user flow rate r. The ratio between control message and reserved data traffic is determined by $\eta = \frac{S}{R \cdot r}$.

Suppose there are N = 10,000 RSVP sessions going over a backbone link. The soft-state refresh interval R is 30 sec, and the combined message size S is 350 bytes[5]. We have $B_{ctrl} \sim$ 0.93 Mb/s. Take r to be 64 kb/s, the typical voice transmission rate. We have $\eta \sim$ 0.146%, which is not a very large portion of the overall reservable link bandwidth. This implies that, when RSVP is used to make explicit bandwidth reservations, the maximum number of sessions on any given link is bound by the reservable link capacity, and the upper bound for control message bandwidth usage is predictable and manageable.

However, there is one exception when dealing with MPLS "label-switched path", where RSVP is used to setup and maintain the connections. Since each RSVP session may or may not be associated with link bandwidth, B_{ctrl} is no longer bounded. On the other hand, MPLS is designed to operate in high-speed networks, where the bandwidth used for exchanging control messages should not be a serious issue.

Nevertheless, efficiently utilizing bandwidth should be one of the important objectives in every signaling protocol design. In Chapter 3, we will introduce a scheme called *staged refresh timers* that can reduce B_{ctrl} by increasing refresh interval R, while improving mes-

[5]Both R and S are the default values specified in RSVP and IntServ. A typical RSVP Path message is 200 bytes, and a RSVP Resv message for Controlled Load service is 150 bytes.

sage delivery reliability. We will also evaluate a mechanism, *summary refresh*, that reduces B_{ctrl} by reducing reservation message size S. Another method is to limit the total number of reservation sessions, N, by introducing *reservation aggregation*. In Chapter 7, we will present a sink-tree based aggregation method for inter-domain reservations that can achieve good bandwidth scaling properties.

2.5.3 Message Processing Cost

Message processing cost is directly related to protocol complexity. Here, we must differentiate the complexity introduced by running resource reservations inside a network from the complexity and the scalability problems that are involved in reservation protocol design and implementation.

Implementing RSVP in a scalable manner is difficult. This is because the protocol is receiver-initiated, multicast-driven and supports shared reservation and complex error recovery algorithms. In particular, receiver-initiated reservation is troublesome, since the routers must develop *efficient* algorithms to map the reservation requests that are installed by the senders to the reservations initiated by the receivers, while taking into the consideration of network topology changes, one-to-many mapping (multicast) and many-to-one mapping (shared reservation). Nevertheless, the IBM RSVP router implementation [65] was able to setup a new RSVP session in 1.1 ms, and process a refresh message in 0.64 ms for a low-cost, low-performance 32 MHz Motorola processor.

By simplifying the reservation sequence on routers, we can drastically reduce processing overhead. For instance, we proposed a lightweight sender-initiated, one-pass reservation protocol, YESSIR. On the same IBM router platform, the YESSIR implementation improved the reservation setup time by 70% over RSVP. It was also shown that with careful implementation and by using basic hashing techniques to manage reservation states, a stand-alone YESSIR implementation can process up to 10,000 flow setups per second (or support up to 300,000 flows per router) on a 700 MHz Pentium PC [66]. We will describe YESSIR's design and implementation in Chapter 6.

2.5.4 State Management Cost

Each individual reservation needs to be managed separately by providers. Improper reservation management can cause service disruption and waste network resources. Management includes tasks such as optimizing and monitoring user traffic. The complexity and the overhead involved in reservation management increase linearly with the number of reservations. Therefore, there can be a scaling problem if there are too many reservations in the network.

To grasp the scope of this issue, we need to understand the maximum number of "states" that providers are capable or willing to handle. A close analogy is the number of routing policies managed by each ISP. Presently, network providers regulate incoming and outgoing transit traffic by setting up routing policies at border routers. A typical routing policy consists of a list of destination networks whose traffic the router is allowed or not allowed to receive. Each policy is derived from the bilateral agreement among neighboring providers. If an ISP does not manage its routing policies properly, it will cause traffic "flooding" and looping in the Internet. The consequence is catastrophic and can result in neighboring providers blocking all traffic coming from and going to the offending ISP. The maximum number of routing policy entries at a large provider's backbone is of the order of 100,000, which is the total number of routes in today's Internet. In reality the actual number of routing policy entries is much smaller due to route aggregation. We believe that the complexity of handling resource reservations is comparable to managing routing polices.

Figure 2-6 plots the total number of end users, networks and routing domains in the entire Internet over the years. Reservations can be made at either per-user, or per-network, or per-domain level. Though reservations may spread out among many routers and networks, it is possible that a transit provider or a border router has to process a large number of reservations.

One important factor that contributes to the large number of reservation states depends on how the signaling protocol is designed. The de facto signaling protocol, RSVP, has a state management problem. To support a RSVP-initiated reservation, each router has to maintain both reservation source and destination information. We shall further investigate

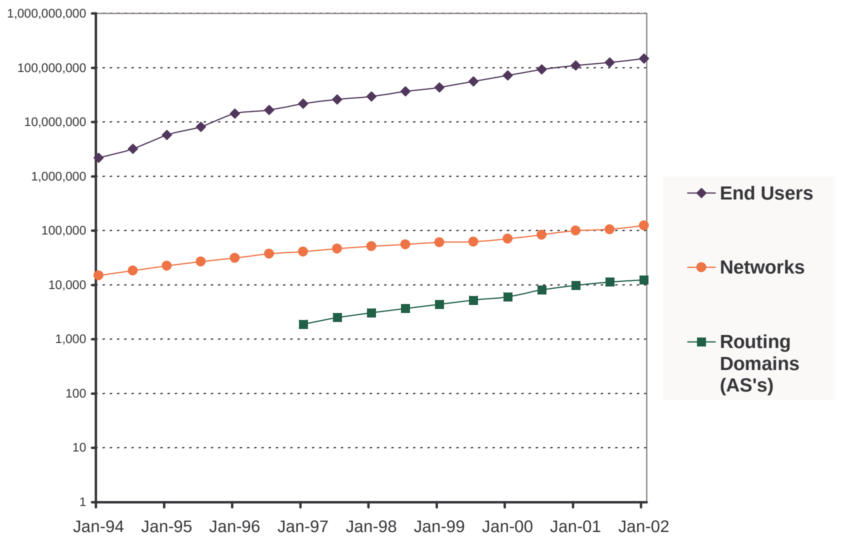


Figure 2-6: The growth of the Internet from 1994 to 2002 [1, 2, 3]. The number of networks was collected by Geoff Huston of Telstra at AS-1221 by monitoring BGP routes. The AS data was collected by Tony Bates from Global Crossing Network. Due to CIDR aggregation, both numbers can differ slightly at different locations inside the network.

the RSVP state management problem in Chapter 4.

In conclusion, we believe that managing reservations efficiently is very critical to network operation, and can be difficult to accomplish. We propose a two-folded solution. First, we need to reduce the number of reservations that network operators have to manage. This can be accomplished by introducing a hierarchical reservation model in the Internet, where the user-level reservations are processed by the users and access networks, while the backbone networks only maintain reservations at network-level or AS-level. The second part of the solution is to design a new reservation protocol that has a good scaling property for the backbone networks. In Chapters 4 and 7, we will further describe the concept of hierarchical reservation and scalable backbone reservation protocols.

2.6 Discussion

From our investigation in Section 2.3, we discovered that the Internet has bandwidth bottleneck links that can cause long-lasting congestion. At the same time, leased line cost has not been reduced sufficiently in a timely manner for many ISP's to deploy high-speed links everywhere in their networks. The continuing Internet expansion has simply aggravated this situation. It is our belief that the applicability of over-provisioning is questionable.

It is possible that over-provisioning is sufficient to reduce packet transaction delay and link congestion *within* some of the large backbones, however, it is not a cost-effective practice for most ISP's. Many providers have been adapting techniques such as web caching and private peering to reduce the cost and the volume of transient traffic,

We cannot predict exactly what will happen to the Internet in the future. In today's fast growing inter-networking environment, network deployment is not about meeting today's traffic demand, instead, it is to anticipate and be ready for new services, new applications and new types of traffic. We believe that it would be a near-sighted argument to rule out the possibility of wide-deployment of inter-active real-time applications (such as voice over IP) in the Internet. Since it is almost impossible to sufficiently over-provision network resources for the future, thus, we have to seek for a better solution that can accommodate network growth at a reasonable cost. If we continue to allow, as we have seen today's Internet, traffic sources to grab resource as they need for free, and not providing traffic isolation and protection to mission-critical data flows, we are bound to experience network congestion and unacceptable end-to-end delays.

Applying resource reservation can bring many benefits to the network. It can provide traffic protection and service guarantees to user flows inside the network, so that the remotely located user populations can have acceptable and consistent communication over the Internet. Also, it can be used as a tool to service providers to prevent unregulated traffic from flooding the network to degrade other user's applications.

We need to be careful with where to apply resource reservation. Simply because reservations can protect user flows from temporary link congestion, it is *not* a tool to compensate for resource inadequacies caused by over-subscription, and sloppy network design. In fact, applying resource reservation in an over-subscribed network can cause service degradation to many users, and aggravate congestion conditions in the network.

We need to separate the concept of resource reservation from the complexity and scalability issues involved in resource reservation protocols. Issues for the former include when and how to apply reservations. Message processing, state maintenance and reservation aggregation are the challenging issues that fall into the latter category, and will be addressed in the rest of this dissertation.

Resource reservation requires states to be installed and maintained at the routers. To support a large number of reservation flows efficiently, reservation protocols must be scalable and relatively simple to process. From our studies of the reservation scalability issues, we argue that the scalability problem is not only an issue about the number of the flows that routers can process, but also an issue about the number of the flows that providers can manage for policing, accounting and billing.

In conclusion, we believe that much future research is needed on understanding network resource manageability.

Chapter 3

Enhancing RSVP Scalability

3.1 Introduction

In this chapter we explore the scaling problems in RSVP state management, and present a simple and backward-compatible solution that can reduce signaling traffic volume inside the network.

3.1.1 RSVP

RSVP (ReSerVation Protocol) [22, 23] was the first widely developed [67, 58, 68, 69] and well studied [70, 71, 72, 73, 74, 75, 76] resource reservation protocol in the Internet. It was originally designed as the signaling protocol to support Integrated Services (IntServ) [30, 31, 32, 33], where end users can trigger RSVP to establish simplex reservation "flows" in the network. Each flow is defined as a source-destination pair, and each destination can be either a unicast user, or a multicast group.

RSVP is a two-pass signaling protocol, that is, it takes two messages, PATH and RESV, to complete a reservation. To setup a reservation between source S and destination D, S first adds a reservation description or *flowspec* in a PATH message. The PATH messages are periodically sent toward D. Each router along the way records the flowspec from S. Upon reception of a PATH message, D adjusts the flowspec to its needs, and puts a modified flowspec in a RESV message. RESV messages are periodically sent toward S along the path traveled by the PATH messages. At each router, a local reconciliation must be performed on the flowspec's from S and D. A local reconciliation process is to merge the flowspec's in such as way that the resulting flowspec can satisfy the reservation requirements from D while within the bound of the original requirements from S. If the router can accommodate the resulting flowspec, a reservation is made and the RESV message is passed on.

During RESV message processing, if a router cannot establish the required reservation, it returns an explicit error message back to D. At the same time, the router has to keep a copy of the failed flowspec from D, and retry the reservation during the next refresh cycle. This latter process is referred as *killer reservation prevention*: since the failed reservation requests are identified and cached, they won't deny service to other requests during the reservation process. The failed flowspecs that routers keep for reservation retries are called *blockade states*.

RSVP uses the underlying routing protocols to discover the reservation path. When a router detects a route change, it sends PATH messages to the new route for the flows whose route has changed. When the downstream routers, situated at the junction of the old and new routes, receives these PATH messages, they immediately sends a RESV message upstream along the new route to setup a new reservation. This process is called *local repair*.

There are two styles of reservations in RSVP: distinct and shared. A distinct reservation, called *fixed filter* (FF), maintains one reservation per source-destination pair. A shared reservation allows multiple sources going to the same destination to share a *single* reservation in the network. RSVP permits a reservation to be shared in two ways: a *shared-explicit* (SE) reservation is to have only explicitly identified sources sharing a reserved flow, whereas a *wildcard filter* (WF) allows a reservation to be shared by all sources.

RSVP uses *soft-state* to maintain reservation states. Each router periodically issues its own PATH and RESV messages to the adjacent routers about the reservations it holds. Reservations will disappear by themselves if they are not refreshed periodically. This avoids orphan reservations and allows reservations to adapt to routing changes, without involvement of the end systems.

RSVP sends its control messages as IP datagrams with no reliability guarantee. It relies on the periodic refresh messages from hosts and routers to handle the occasional loss of a PATH or RESV message. Each RSVP host or router maintains a cleanup timer. A state is deleted if no refresh messages arrive before the expiration of a cleanup timeout interval.

To remove a reservation from the network, users can either stop sending the PATH and RESV refresh messages, or issue explicit teardown messages, PATHTEAR and RESVTEAR.

3.1.2 Problems with Soft-State Refresh

The use of *soft states* in RSVP can result in both reliability and scaling problems. The reliability problem occurs when RSVP messages are lost during transmission. Packet losses in the current Internet can be frequent on some links, unfortunately. In one extreme case, it was measured in 1996 [77] that the packet loss rate in the Internet multicast backbone (Mbone) [78] was approximately 1-2% on average, and could occasionally reach 20% or more on congested links. The existing RSVP message delivery mechanism will not work well in such an environment. For example, when a user tries to make a reservation over the network, if the *first* reservation request (RESV) is lost due to congestion, it will not be retransmitted over the congested link until the next refresh cycle. The default refresh interval is 30 seconds.

Thus, the first few seconds of, say, a multimedia flow may experience degraded quality of service as packets are carried on a best-effort basis rather than as a reserved flow. Unfortunately, packet loss is more likely to delay reservations just when needed most, *i.e.*, when packet loss rates for best-effort service are high.

Another problem related to reliability is that RSVP does not retransmit tear-down messages. If, for example, a user tries to remove a reservation, and the message (RESVTEAR) is lost, the reservation will remain in place until it times out, by default after 90 seconds. If holding a reservation incurs costs, the user will have to pay for the extra time that has been spent waiting for the reservation to time out. Also, network resources are used inefficiently. Network providers will have to account for this uncertainty in their billing policies.

The scaling problem is linked to the resource requirements in terms of bandwidth usage and processing overhead. The resource requirements increase proportionally with the number of RSVP sessions. Each session requires the generation, transmission, reception and processing of PATH and RESV messages for each refresh period. Supporting a large number of sessions, and the corresponding volume of refresh messages, presents a scaling problem. Specifically, the bandwidth usage, as expected, grows linearly with respect to the number of sessions. Depending on implementation, the processing overhead is, at best, linear with respect to the total number of sessions, as shown in [69].

One way to improve the reliability of RSVP signaling is to decrease the refresh period on all network nodes. This increases the probability that state will be installed in the face of message loss, but at the cost of increasing refresh message volume and associated processing requirements and aggravates the scaling problem. On the other hand, a simple way to reduce the effect of scaling problem in RSVP is to increase the refresh period on all network nodes. However, this increases the time it takes to synchronize reservation state.

In the rest of the chapter, we present a simple RSVP extension, called *staged refresh timer*, that provides a mechanism to deliver RSVP messages faster and more reliably, and reduce the number of refreshes among network nodes, contributing to protocol scalability. We will also briefly describe two other mechanisms, *message bundling* and *summary refresh*, that can further reduce the processing cost of refresh messages.

3.1.3 Terminology

We use the following terms during our description of the operation.

Sending and receiving nodes: A *sending node* is a router or host that generates RSVP messages. A *receiving node* is defined as the RSVP router or host that is one hop away from a sending node. In a shared-media or non-broadcast multiple access (NBMA) network such as an ATM subnet, a sending node may have multiple receiving nodes. In some cases, not all routers between sending and receiving nodes implement RSVP. We refer to these networks as *non-RSVP clouds*.

Trigger and refresh message: In RSVP, control traffic can be categorized into two types: trigger and refresh messages. *Trigger messages* are generated by an RSVP host or a router due to state changes. Such state changes include the initiation of a new state, a route change that altered the reservation paths, or a reservation modification by a downstream router. PATH, RESV, PATHTEAR and RESVTEAR serve as RSVP trigger messages.

Refresh messages, on the other hand, contain replicated state information generated by a router to maintain state. As indicated in the introduction, RSVP periodically refreshes state for robustness. For instance, if the RSVP process on a router crashes and resets, it loses all RSVP state information. However, since its neighbor routers send copies of RSVP state information periodically, the router can recover the lost states within one refresh interval. A refresh message can be either a PATH or RESV message.

3.2 RSVP Staged Refresh Timer Extension

A common practice in supporting message delivery reliability is to use some type of feedback or acknowledgement mechanism between senders and receivers. However, using feedback between RSVP end-users, as has been proposed in [79], will not work unless we significantly change the original protocol. This is because RSVP soft states are managed hop-by-hop, and no network entities other than the node that sent the original refresh message can retransmit a refresh message. In other words, end users have no control over the refresh message delivery process in the network. They cannot accelerate delivery by simply retransmitting RSVP messages. Hence, we conclude that *the only way to accomplish reliable RSVP message delivery is to rely on some hop-by-hop feedback mechanism.*

As mentioned in Section 3.1.1, in case of network failure, RSVP *local repair* procedure allows the routers to quickly identify and rescue the affected reservations. This implies that once reservation states are installed, there is no reason for RSVP to frequently update router reservation states. If the RSVP process is reasonably reliable, refresh messages are more of a safety measure than actually needed for network operation and can thus be sent very infrequently if at all. By doing so, we can greatly reduce the traffic and processing impact of RSVP messages and make RSVP signaling at least as efficient as circuit-switched setup protocols. However, this requires that trigger messages are delivered reliably.

In this section, we will describe a reliable trigger message delivery mechanism based on the above observation.

3.2.1 Outline of Operation

We propose the following feedback mechanism for RSVP trigger message delivery: When sending an RSVP trigger message, a node inserts a new echo-request flag into the RSVP common header of the message. Upon receipt, a receiving node acknowledges the arrival of the message by sending back an echo-reply, which is a new RSVP message type. When the sending node receives this echo-reply for a PATH or RESV message, it will automatically scale back the refresh rate for these messages for the flow. If the trigger message was a flow tear-down, no more tear-down messages are sent, just as in the current RSVP specification. Until the echo reply is received, the sending node will retransmit the trigger message. The interval between retransmissions is governed by a *staged refresh timer*. The staged refresh timer starts at a small interval which increases exponentially until it reaches a threshold. From that point on, the sending node will use a fixed timer to refresh PATH and RESV messages and stop re-transmitting tear-down messages. This mechanism is designed so that the message load is only slightly larger than in the current specification even if a node does not support this staged refresh timer.

The proposed mechanism requires several minor modifications to the current version of RSVP: a new bit is defined in the flag field of the RSVP common header, a new message, ACK, and a new object MESSAGE_ID, is defined for echo-request and echo-reply [80].

3.2.2 Time Parameters

The new extension makes the use of the following time parameters:

Fast refresh interval $\mathcal{R}_f$: $\mathcal{R}_f$ is the initial retransmission interval for trigger messages. After sending the message for the first time, the sending node will schedule a re-transmission after $\mathcal{R}_f$ seconds. The value of $\mathcal{R}_f$ could be as small as the round trip time (RTT) between a sending and a receiving node, if known. Unless a node knows that all receiving nodes support echo-replies, a slightly larger value of, for example, 1 second is suggested.

Slow refresh interval $\mathcal{R}_s$: The sending node retransmits with this interval after it has determined that all of its receiving nodes[1] support the RSVP echo-reply. To manage the number of unnecessary refreshes in a stable network, $\mathcal{R}_s$ can be set to a large value. The value of $\mathcal{R}_s$ can be set for each egress interface. We choose 15 minutes as the default value.

Increment value Δ: Δ governs the speed with which the sender increases the refresh interval. The ratio of two successive refresh intervals is $(1 + \Delta)$. We arbitrarily set Δ to 0.30, which is also the same value as the Slew.Max parameter that has been defined in RSVP to increase the retransmission and timeout interval for long-lived flows using local repairs.

Fixed refresh interval $\mathcal{R}_c$: A node retransmits the trigger message with the interval $(1 + \Delta)^i \mathcal{R}_f$ until the refresh interval reaches the fixed refresh interval $\mathcal{R}_c$ or an echo reply has been received. If no reply has been received, the node continues to retransmit refreshes every $\mathcal{R}_c$ seconds. We choose a value for $\mathcal{R}_c$ of 30 seconds, the same value as the refresh interval in the current RSVP specification.

3.2.3 Staged Refresh Timer Algorithm

After a sending node transmits a trigger message, it will immediately schedule a retransmission after $\mathcal{R}_f$ seconds. If it receives echo-replies, the sending node will change the refresh interval to $\mathcal{R}_s$. Otherwise, it will retransmit the message after $(1 + \Delta)\mathcal{R}_f$ seconds. The staged retransmission will continue until either echo-replies are received, or the refresh interval has been increased to $\mathcal{R}_c$.

The refresh interval for each refresh cycle i, R_i, can be described as:

$$R_i = \begin{cases} (1+\Delta)^i \, \mathcal{R}_f & \text{if no reply and } i \leq I \\ \mathcal{R}_c & \text{if no reply and } i > I \\ \mathcal{R}_s & \text{after echo reply} \end{cases}$$

[1]In case of multicast, in a RSVP session, there could be multiple receiving nodes for a sending node.

where

$$I = \frac{\log \frac{\mathcal{R}_c}{\mathcal{R}_f}}{\log(1+\Delta)}.$$

I is the number of retransmission before the interval reaches the fixed refresh interval $\mathcal{R}_c$.

The implementation of staged refresh is quite simple. The router algorithm operates on two sets of parameters. A set of "global" timing variables consists of $\mathcal{R}_f$, $\mathcal{R}_s$, $\mathcal{R}_c$ and Δ. Each reservation state, k, needs to maintain the following attributes:

k.refresh-timer	the state refresh timer;
k.cleanup-timer	the state expiration timer;
k.type	reservation state type: either active or tear-Down.

k.refresh-timer governs when to send a refresh message. k.cleanup-timer is reset to zero upon receiving of a refresh message. Figure 3-1 shows the router's algorithm at sending nodes. We only show the procedures that are relevant to RSVP timer management. As shown in the figure, if the value of $\mathcal{R}_f\ (1+\Delta)^i$ exceeds $\mathcal{R}_c$ after i tries, we will switch the refresh timer to $\mathcal{R}_c$.

In a network that has many RSVP reservations, routers may have to use a large number of timers to support the staged refresh timer algorithm. This can be a problem for some router operating systems, since conventional timers consume a great deal of processing power. To overcome this problem, we can implement various hashed or hierarchical timing wheels, such as the one being described in [81], to manipulate a large number of soft-state timers with a handful of *real* OS timers.

In an implementation, the staged refresh timer mechanism can be based on several timing wheels, one for each $\mathcal{R}_i$. At message processing time, each reservation state is inserted into one of the timing wheels. Each timing wheel is driven by a periodic timer provided by the router's OS.

Periodic process synchronization is generally considered to be harmful [82]. To avoid the synchronization due to RSVP periodic refreshes, all the timers need to be randomized during the implementation.

```
1: on receiving a trigger message:
        create the corresponding state $k$;
        if the message is a tear-down message
                $k$.type $\leftarrow$ Tear-Down;
                remove $k$'s reserved resource;
        else
                $k$.type $\leftarrow$ Active;
        $k$.refresh-timer $\leftarrow \mathcal{R}_f$;
        forward the trigger message toward the destination.

2: on receiving an echo-reply for $k$:
        if $k$.type $=$ Tear-Down
                delete $k$.
        else
                $k$.refresh-timer $\leftarrow \mathcal{R}_s$;

3: on expiration of $k$.refresh-timer:
        if $k$.refresh-timer $< \mathcal{R}_c$
                if $\mathcal{R}_c < (k.\text{refresh-timer}) \cdot (1 + \Delta)$
                        $k$.refresh-timer $\leftarrow \mathcal{R}_c$;
                else
                        $k$.refresh-timer $\leftarrow (k.\text{refresh-timer}) \cdot (1 + \Delta)$;
                construct the corresponding trigger message for $k$;
                retransmit the trigger message;
        else if  $k$.refresh-timer $= \mathcal{R}_c$ and $k$.type $=$ Tear-Down
                delete $k$;
        else
                construct the corresponding refresh message for $k$;
                send out the refresh message;
```

Figure 3-1: Staged refresh timer at a sending node - router algorithm

Figure 3-2 compares various refresh mechanisms. The figure shows that the number of refreshes is greatly reduced if *both* sending and receiving node support the new extension. If the receiving nodes do not reply to a trigger message, the sending node generates several refresh messages until the refresh interval converges to the fixed refresh interval $\mathcal{R}_c$. While incurring additional overhead, these retransmission increase the likelihood that the

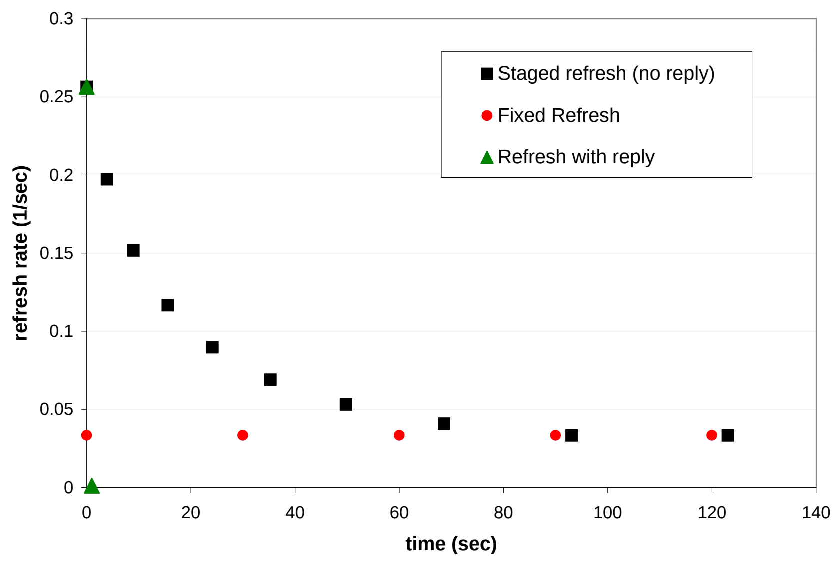


Figure 3-2: Comparison of RSVP refresh rates for different mechanisms: ($\mathcal{R}_f = 3$ s, $\mathcal{R}_s = 15$ min, $\mathcal{R}_c = 30$ s, $\Delta = 0.3$)

reservation state will be established even in a lossy network. In the figure, the sending node transmits $I = 7$ messages before reaching the refresh interval $\mathcal{R}_c$. With a larger value of Δ, $\mathcal{R}_f$ could be decreased to accelerate state establishment.

3.2.4 Protocol Design Considerations

Backward Compatibility Backward compatibility is one of the main objectives in our design. One cannot assume that both sending and receiving RSVP nodes will support the extension simultaneously. To achieve this goal, we have designed the protocol extension in such a way that much of the message manipulation and processing are done at sending nodes, while making no assumptions about the capability of receiving nodes.

In the current RSVP specification, sending nodes refresh the soft states with fixed timers. In our design, sending nodes rely on echo request/reply mechanism to “learn” about the status of receiving nodes. If a sending node does not receive echo replies from the receiving nodes after several tries, it will assume the receiving nodes do not support the new extension, and switch its refresh interval to a fixed value. The RSVP operation is not affected at the receiving nodes.

Computing Cleanup Timeout Values Each RSVP PATH and RESV message carries a refresh interval in its `TIME_VALUES` object. Receiver nodes use the refresh interval to compute the cleanup timeout interval that governs the lifetime of reservation state that has not been refreshed. Generally, the cleanup timeout interval is a small multiple of the refresh interval. As suggested in the RSVP specification, we set the cleanup timeout value to be 3 refresh intervals by default.

In the staged refresh timer design, a sending node initially places the slow refresh timer, $\mathcal{R}_s$, in the PATH or RESV message. For the receiving nodes that do not support the new extension, the sending node will insert $\mathcal{R}_c$ in the refresh messages after the actual refresh interval has been increased to $\mathcal{R}_c$. If the receiving nodes do support the new extension, they will set the cleanup timeout interval based on $\mathcal{R}_s$.

Figure 3-3 summarizes the algorithm that processes messages on receiving nodes. Note that the receiving nodes do not need any of the staged refresh timer parameters. The nodes simply send an echo-reply back to the sending node after receiving a trigger message.

Handling of Tear-Down Messages RSVP uses PATHTEAR and RESVTEAR messages to tear down path and reservation states, respectively. According to the current specification, sending nodes only generate one tear-down message per flow. If the message is accidentally dropped along the way, the reserved resource will not be released until the cleanup timer expires. However, receiving duplicate tear-down messages at a receiving node should not impact the operation of RSVP in a proper implementation.

In our RSVP extension, we have altered the processing rules for tear-down messages at the sending node. Instead of deleting the state after a tear-down message is sent, a sending node will release all resource allocated to the state, and mark the state as *closing*. The state information is saved for message retransmission. The entire state information will be removed when echo-replies are received, or when the sending node realizes that the

```
1: on receiving a *trigger* message carrying refresh value $R$:
       find/create the corresponding state $k$;
       **if** the message is a tear-down message
           free $k$'s reserved resource;
           delete $k$;
       **else**
           $k$.cleanup-timer $\leftarrow 3 \cdot R$;
       **if** support the staged refresh extension
           reply an echo-reply back to the sending node.

2: on receiving a *refresh* message for $k$ carrying refresh value $R$:
       $k$.cleanup-timer $\leftarrow 3 \cdot R$;

3: on expiration of $k$.cleanup-timer:
       free $k$'s reserved resource;
       delete $k$.
```

Figure 3-3: Staged refresh timer at a receiving node - router algorithm

receiving nodes do not support the extension.

Operation in an NBMA Environment For a multicast RSVP session in a non-broadcast multiple access (NBMA) network (such as ATM), a sending node may not know the total number of receiving nodes for a PATH or PATHTEAR message at an egress interface. Therefore, a sending node cannot simply switch to the longer refresh timer R_s based on having received echo-replies.

For example, as shown in Figure 3-4, if the receiving node R3 does not support the new RSVP extension, the sending node S should not change to the longer refresh interval $\mathcal{R}_s$, even though it has received echo-replies from R1 and R2.

In this case, a sending node has two alternatives:

- It can query a local database such as the ARP or MARS server [83, 84] to find out the exact number of next-hop receivers. It then switches to a longer refresh interval after receiving echo-replies from all receiving nodes.

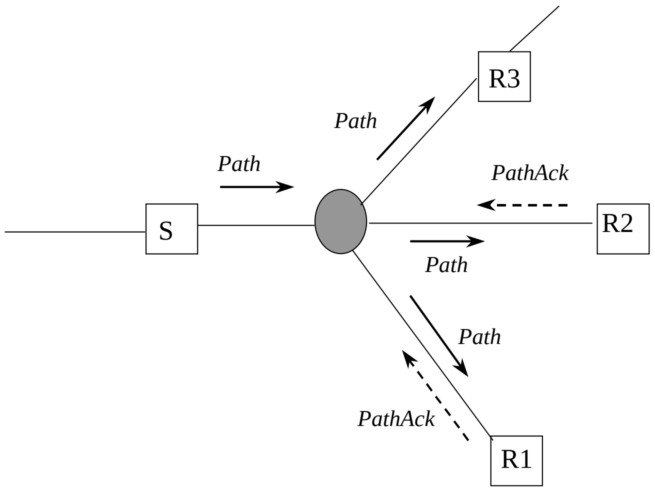


Figure 3-4: Example: PATH messages in an NBMA network.

- Since PATH messages are mainly used for traffic advertisement purposes, the sending node may not need to use staged refresh timers for PATH messages. In an NBMA network, the staged refresh time mechanism would only make sense for the message delivery of RESV, RESVTEAR and PATHTEAR messages.

In case of PATHTEAR message, a sending node always knows all the receiving nodes that have made reservations during the RESV message processing time.

As shown in Figure 3-5, R1, R2 and R3 are the receiving nodes to S. Initially, the sender S had the reservation state information for receiving node R1 and R2. Since R3 did not make any reservation, S would not know the existence of R3 from its RSVP database. After sending the first PATHTEAR message, S will retransmit the message until it has received echo replies from R1 and R2, and then stop generating PATHTEAR messages.

3.3 Evaluation

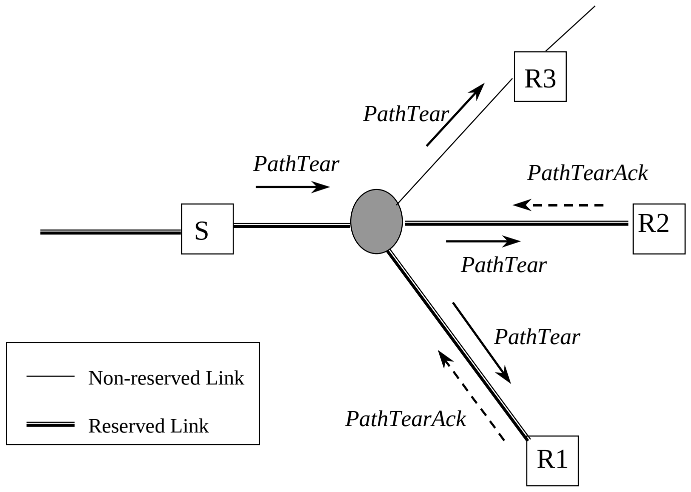


Figure 3-5: PATHTEAR message in an NBMA network

	60 s	60 min
Fixed refresh	300	18,000
Slewed refresh	300	1,950
Staged refresh (no reply)	900	18,600
Staged refresh (with reply)	300	900

Figure 3-6: Protocol overhead (bytes sent) for setting up a new reservation session in the first 60 seconds and 60 minutes for different RSVP timer mechanisms: $(\mathcal{R}_f = 3 \text{ s}, \mathcal{R}_s = 15 \text{ min}, \mathcal{R}_c = 30 \text{ s}, \Delta = 0.3)$

Reduced Protocol Overhead When RSVP trigger messages have been acknowledged by echo-replies, the soft state refresh frequency is reduced. Figure 3-6 shows the protocol overhead for a single RSVP flow, using the same parameters as before. The RSVP message size is 150 bytes, which is the size of a regular RESV message. We also compare this to the mechanism described in [23, p. 57], which simply increases the refresh interval by a slew factor (here, 0.3). As we have discussed previously, increasing or decreasing the refresh interval can have undesirable reliability and scalability effects to RSVP. If the receiving node supports the staged timer extension, the number of bytes to be transmitted in an hour is only 900 bytes. Even if the receiving node does not support the new extension, the amount of data being transmitted over an hour is nearly the same compared with the fixed refresh timer case.

For a link that requires to manage thousands of RSVP flows, protocol overhead reduction is clearly an advantage over the fixed refresh timer.

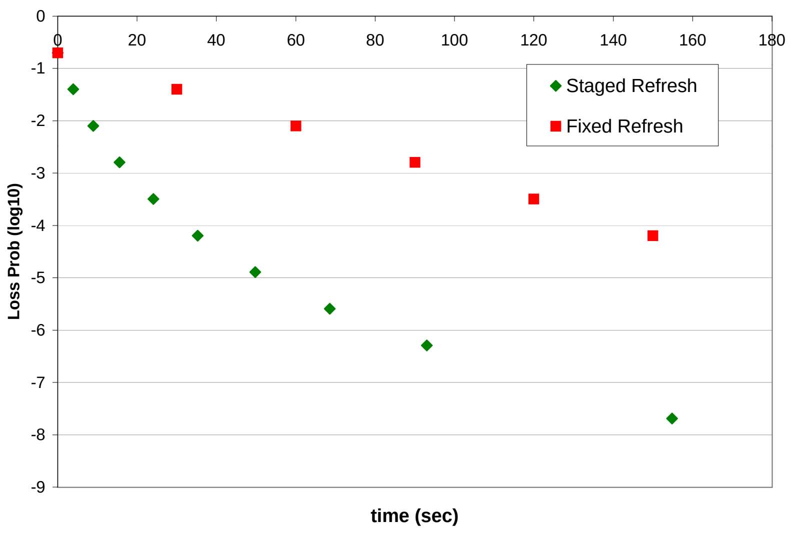


Figure 3-7: Comparison of message loss probability as a function of time for fixed and staged refresh timers.

Reduced Message Loss Probability Adding feedback mechanism in RSVP message delivery reduces the message loss probability. Figure 3.3 shows the cumulative loss probability for both fixed and staged refresh mechanism. We assume the message loss probability for a single message is 20% [77]. In the example, four refreshes are sent within the first 30 seconds if a staged refresh timer is used, compared with only one refresh with the fixed refresh timer. The probability that no reservation is established after half a minute is reduced to the neighborhood of $3 \cdot 10^{-4}$ compared with 4% with the current fixed timer. For a loss rate of 2%, the failure probabilities are $3 \cdot 10^{-9}$ and $4 \cdot 10^{-4}$, respectively.

3.4 Related Work

There have been several other proposals [80, 85, 86] in recent years on how to improve scalability and reliability in soft-state-driven protocols.

The mechanisms introduced in [85, 86] require both sending and receiving nodes to operate a set of new refresh reduction algorithms simultaneously. Since RSVP (in the form of MPLS) has been deployed in the backbone networks, such as UUnet, AT&T, and Level-3, both proposals present a backward compatibility problem and may cause deployment difficulties.

Summary refresh and *message bundling* [80] are two simple and backward compatible RSVP extensions that can reduce refresh message processing overhead on routers.

3.4.1 Summary Refresh

The idea of *summary refresh* is the following: RSVP refresh messages have the same format as RSVP trigger messages. Given the messages can be quite large, it is a waste of link bandwidth and processing power to retransmit the same identical information at each refresh cycle. Summary refresh is to simply assign an identification number to each RSVP refresh message at reservation setup time. At each refresh cycle, the sending node sends those identification numbers in one datagram. However, if the reservations have been modified, or the routers have been rebooted, the identification numbers will have to be re-assigned.

3.5 Message Bundling

Message bundling is to pack several RSVP messages between two neighboring nodes into a single IP datagram. The goal is to reduce the total number of RSVP messages that routers have to process. This solution is particularly useful to relive the processing burden on routers that are running a UNIX operating system, since the socket interface has always been a processing bottleneck. Although BSD, a popular version of UNIX, has been redesigned over the years to improve virtual memory and I/O system interface [87], processing overhead between kernel and user space through sockets remains significant.

To demonstrate the problem, we ran a simple test on an Intel Celeron 500 MHz PC running FreeBSD 3.4. This PC was connected to two other PC's through 10 Mb/s Ethernet interfaces; we confirmed that it could route traffic between the interfaces at media speed. Our goal is to measure the message processing time in both kernel and user space. RSVP PATH messages are generally encapsulated with the IP Router Alert option [88, 89]. Since the processing time should be the same for all IP option types, and we want to have the flexibility to evaluate the performance variation base on packet size, we used the *ping -R* command to generate IP record route option packets. We used a modified FreeBSD kernel that can intercept IP option packets from the wire and direct them to a user space handler quickly through raw sockets [90]. In user space, we implemented a system function call, *ipodump*, to reinject IP option packets back into the network immediately after intercepting them. Two simple timing checks were added in the kernel, one at the beginning of the IP input, the other at the end of IP output routine. Using this, we can measure the processing delay at the socket interface.

The results of our measurements are shown in Figure 3.5. The processing delay does not depend significantly on the packet size. Out of the 16 sets of data we collected, the delay for processing 1-byte ICMP packets with IP record route option was 116.4 μsec, compared with 132.6 μs for 1400-byte packets, *i.e.*, only a 12% increase. We believe this is due to the efficient memory management in BSD.

However, the average packet processing time in user space, including reading from and writing to sockets, was 46.28 μs. In comparison, an intercepted packet spent 124.64 μs on average traversing kernel and user space. That is, the BSD socket interface contributed as much as 78.36 μs or 62.87% of the process!

The test results thus imply the following: *for BSD-based routers, increasing message size does not impact the performance very much, but reducing the number of messages on the wire can greatly reduce the processing overhead.*

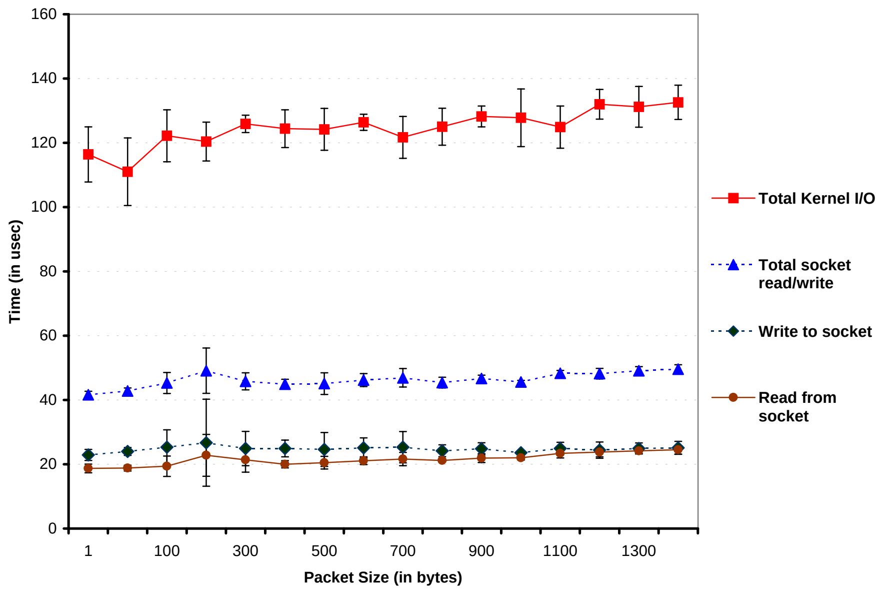


Figure 3-8: Comparing kernel and user processing overhead

3.6 Discussion

The RSVP message delivery mechanism requires some degree of reliability guarantee to make RSVP useful for applications. One way of improving reliability is to reserve some minimal bandwidth for RSVP messages to protect them from congestion losses, as suggested in the RSVP specification. However, this may require additional functionality at both sending and receiving nodes and does not help if RSVP messages have to traverse non-RSVP clouds. It is also not clear how this can be achieved in a backward-compatible manner. Most importantly, since there is no way of knowing the exact bandwidth that is required to carry RSVP control traffic, allocating a specific level of link bandwidth may not prevent RSVP message dropping.

We have presented a mechanism called *staged refresh timer* that enhances the current RSVP message delivery and is completely backward compatible. Staged refresh timers are easy to add to RSVP-capable routers and host implementations and save both processing and bandwidth overhead. The staged refresh timer mechanism is an example of state management that falls somewhere between “classical” handshake-based reliability as found in B-ISDN (Q.931) signaling, for example, and purely timer-based soft-state protocols such as the original RSVP proposal [22], PIM [91], RTP [24], SRM [92], IGMP [93] and so forth.

Since only neighboring routers are involved in the reliability mechanism described here, the routers can easily estimate round-trip times, thus further tightening the retransmission interval, if desired.

Staged refresh timer, message bundling and *summary refresh* have been standardized by IETF and documented in RFC2961. They have been implemented on Juniper routers, and deployed in some of the large backbone networks. It was estimated that, in a stable network environment, the combination of all three mechanisms can reduce the total RSVP processing overhead by as much as factor of 12.

Chapter 4

The Hierarchical Reservation Model

In Chapter 2, we concluded that the scalability problem in resource reservation may *not* be due to memory constraints or per-flow queuing at the data forwarding layer, but rather depend on the total number of reservation flows that routers have to maintain, and the total number of reservations that network providers have to manage.

While the techniques described in Chapter 3 improve signaling scalability and control message delivery, it is unlikely that we can use a single signaling protocol for all the users throughout the Internet, considering there could be millions of reserved sessions from end-users and VPN connections spanning multiple networks.

In this chapter, we define a new resource reservation architecture, the *hierarchical reservation model*, that defines how and where the reservations should be used by end-users and network providers to satisfy the functional and performance requirements of a wide variety of services and applications.

Before go on to discuss the new architecture, we need to first understand the scope of the scalability problem. Namely,

- What is the upper bound of the number of reservations in today's Internet?
- What is the upper bound of the number of reservations that routers can handle?
- Are the existing signaling mechanisms adequate for the job?

4.1 Reservation Scalability Revisited

4.1.1 The State Management Problem

In most reservation signaling protocols known today, including RSVP and ST-II [94], each reservation is uniquely defined as the combination of a *source* and a *destination*. Generally, a source or a destination can be represented by an IP address, or a transport-layer port number, or the protocol type, or the combination of all three. Network routers must maintain both source and destination for each flow during its duration.

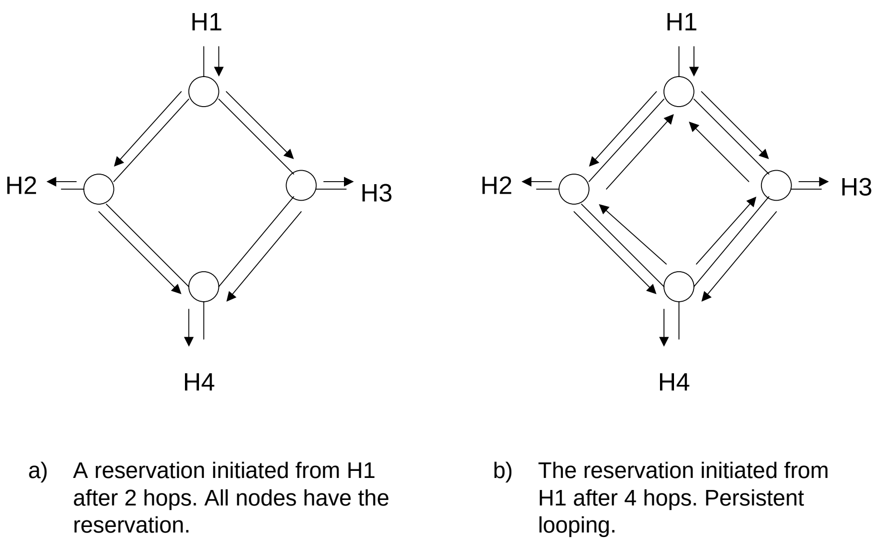

H1
H2
H3
H4
a) A reservation initiated from H1 after 2 hops. All nodes have the reservation.
H1
H2
H3
H4
b) The reservation initiated from H1 after 4 hops. Persistent looping.

Figure 4-1: Reservation looping problem in multicast: $H1, H2, H3$ and $H4$ belong to the same multicast group G, and don't keep track of reservation source information.

The reason behind maintaining both source and destination at routers is to prevent persistent looping caused by *multicast*. For example, in a 4-node network shown in Figure 4-1, all nodes belong to the same multicast group G. $H1$ sends an RSVP reservation request (RESV) message to the group. If the routers do not maintain the reservation source information, each router forwards reservation request in both directions, resulting in the loop shown in Figure 4-1-(b). Even after $H1$ stops refreshing its reservation, the loop persists because each router in the loop has a child that refreshes the reservation. The loop is only broken when any member of the group initiates a tear-down message to explicitly remove the reservation.

To correct the problem, routers have to store both source and destination for each reservation. Upon receiving a reservation request, any router can identify the sender tree that the message is following. It can avoid looping by simply obeying the rule that *a reservation state that is received through a particular interface must never be forwarded out the same interface*. In Figure 4-1-(a), when $H4$ receives a reservation request ($H1$, G) from $H2$ and $H3$, respectively, it will not forward the reservation message back toward $H3$ and $H2$, respectively, thus no loop can be formed.

Routers do not need to store the source information when multicast routing uses a single, shared tree for all senders, in which case, all receivers share the same loop-free branch to each sender. Core Base Tree (CBT) [95] is such a multicast routing protocol, where all multicast senders share the same routing tree to the receivers. On the other hand, since PIM [91] allows multicast senders to be on both shared (RP) trees as well as shortest path (SPT) trees, reservations may still run into looping condition when using SPT trees.

Another exception is sender-initiated reservation protocols, where reservation messages are forwarded on exactly the same path as IP data packets. There should be no reservation loops except perhaps for transient routing loops. When there is a transient multicast routing loop, at routers, the routing protocols can terminate the loops eventually, and notify the reservation module to fix all affected reservations. But the notifications have to rely on source and destination information to identify each reservation.

In conclusion, many of the today's reservation protocols inherently have the problem of maintaining a large number of states in the routers, which can potentially escalate to the reservation state "explosion" situation that we will describe below.

4.1.2 Reservation Granularity

Since resource reservation is not yet widely used, we have to extrapolate the likely volume of reservation state from other observations. Figure 2-6 has shown the network growth rate in the last 7 years. It's here to re-emphasis the network growth trend. As of January 2002, there were more than 100,000,000 end-users, more than 100,000 networks (advertised through BGP routing protocol) in the Internet backbone, and more than 10,000 AS's (Autonomous Systems[1]). Assume we use RSVP to setup reservations in the Internet. Because of the state overhead problem of RSVP, the upper bound for the number of reservations Internet-wide can be very large.

From the figure, we can see that future network growth poses an even more serious problem: the number of end-users, networks and carriers is increasing at a rate between 60% to 80% on average every 12 months in the past five years. This growth pattern is comparable to the Moore's Law, which a well-known metric to describe the technological growth in hardware. The Moore's Law predicts that hardware (including clock rates and memory sizes) doubles every 18 months, and a growth exceeding this implies costs will grow rapidly. To support the fast growing Internet size, network providers must gradually start to upgrade or replace the routers and the switches inside the network. To accommodate routing table explosion, and the processing overhead in inter-domain routing communication, the entire routing architecture needs to be re-evaluated. At the same time, to efficiently support a large number of reservations in the network, we need to overcome the state management problem introduced by conventional signaling protocols.

To understand the maximum number of reservations that routers have to handle, we have collected a 90-second traffic trace [96] from the MAE-West network access point (NAP). We categorized about 3 million IP packet headers from June 1, 1999, according to their transport-layer port, IP address, IP network prefix and BGP Autonomous System (AS) number. Table 4.1 shows the results; for example, if we use RSVP, the total number of reservations can range from 20,857 if we reserve source-destination AS pairs, or up to

[1]This gives a rough estimation on the total number of carriers in the Internet, though some of the large carriers may own multiple AS numbers.

Granularity	flow discriminators	flows
Application	source address, port	143,243
	dest. address, port, proto.	208,559
	5-tuple	339,245
IP Host	source address	56,935
	dest. address	40,538
	source-dest. pairs	131,009
Network	source network	13,917
	dest. network	20,887
	source-dest. pairs	79,786
AS	source AS	2,244
	dest. AS	2,891
	source-dest. pairs	20,857

Table 4.1: Flows and aggregations based on a 90 s packet trace from MAE-West

339,245 if every flow identified by a unique 5-tuple[2] gets its own reservation.

From the table, we observe that there are about 21,000 unique source-destination AS pairs which a backbone router should be able to handle. However, this number is artificially low due to the small 90-second window. Over the span of a month (May 1999), MAE-West saw 4,908 unique source AS's, 5,001 unique destination AS's and 7,900,362 unique AS pairs, out of the 25 million possible combinations[3]. The measurement result is consistent with Bates's statistics [3]. The table also indicates that the number of source and destination AS's and networks is relatively small. Bates' statistics [3] show that there were approximately 5,000 autonomous systems and fewer than 60,000 network prefixes in the Internet in June 1999.

In conclusion, *if we set up reservations based on either source or destination AS's or network prefixes, we can readily keep the reservation count at levels sustainable by today's routers and providers.*

[2]5-tuple is the combination of source and destination addresses and port numbers, and the protocol type

[3]The AS information was collected on June 10, 1999 and analyzed by Sean McCreary of NLANR/CAIDA.

4.1.3 What to Reserve?

It has been argued that since network link bandwidth is finite, therefore it is unlikely that a link would see thousands of reservations.

This is supported by our observation on the MAE-West link. We have collected eight packet header traces from MAE West. The traces were collected three hours apart on June 1, 1999. Each trace comprises 90 seconds of all traffic at the NAP and contains about 33 million packet entries. We collected the number of bytes for each pair of source and destination route prefix, and for each destination route prefix separately. We sorted the data into five categories: fewer than 50 B/s, 50–500 B/s, 500–2000 B/s, 2000–8000 B/s, and greater than 8000 B/s. Figure 4-2 plots the distribution of flows by bandwidth.

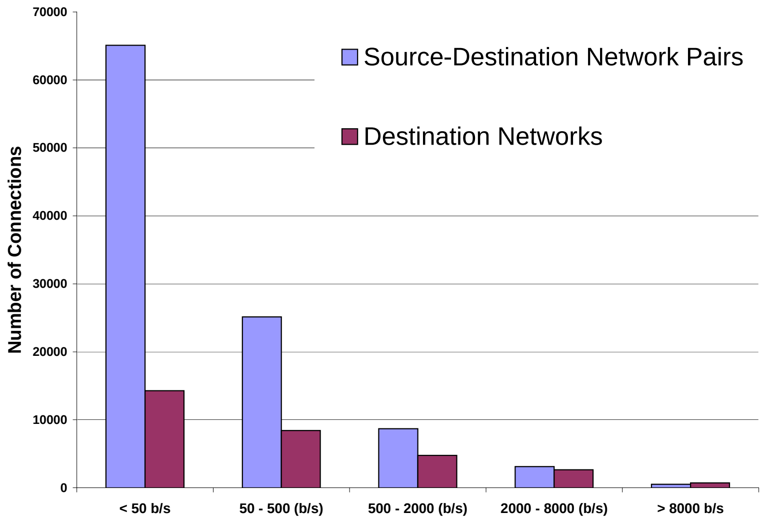


Figure 4-2: Distribution of connection by bandwidth.

One interesting observation from the trace is the number of source-destination pairs and destination-only “flows”. In Figure 4-2, the number of destination-only flows is significantly less than the number of source-destination pairs. We have extracted the data and presented the ratio in Figure 4-3. For small flows (*i.e.*, average bandwidth less than 50

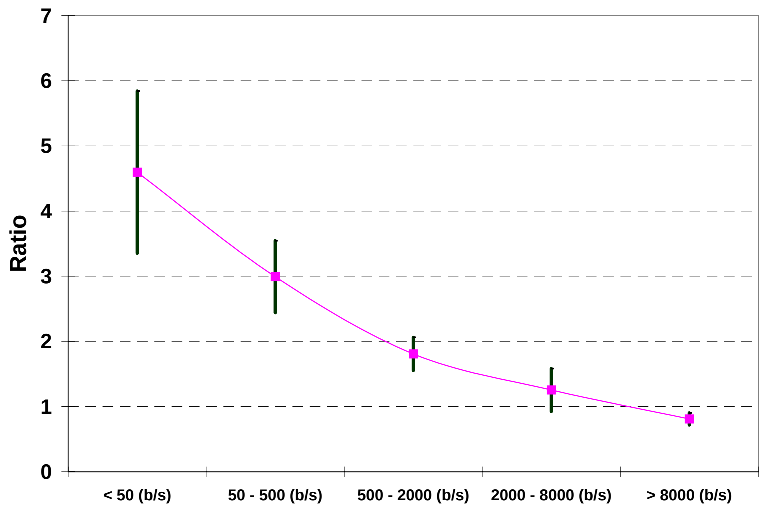


Figure 4-3: Source-destination-pair to destination-only ratio with standard deviation

B/s), there are more than four times as many source-destination-pair flows compared to the flows based on the destination prefix alone. Once again, we see evidence that if we can work around the potential state“explosion” problem in reservation protocols, and use a destination-based signaling mechanism, we can reduce the number of flows at routers.

Most of packets belong to the small-flow category: 63.5% for source-destination pair, and 46.2% for destination-only. Only 3621 (3.5%) of the source-destination pairs and 1296 (10.9%) of the destinations have an average bit rate over 2000 b/s. Interestingly, there are more above-8000 b/s destination-only flows (719) than source-destination flows (516). We suspect that these destinations may be the large Web servers.

The measurement seems to imply that *if reservations are made only for high-volume traffic sessions, then such flows are rare enough that reservation protocol scalability is not an issue.*

However, this argument may not be true in the long run. With the deployment of IP telephony, the number of real-time interactive traffic flows is likely to increase. At the same time, network bandwidth will likely rapidly increase due to the deployment of optical switches into the Internet backbone. Both factors may result in a large number of bandwidth reservations in the network.

More importantly, to provide service differentiation and VPN features to customers, the network providers have been using reservation signaling protocols [97] to set up MPLS LSP (Label Switched Path) tunnels. MPLS LSP's are very similar to ATM VP's. Each LSP connects two network nodes and the connection in turn carries datagram traffic. Users can establish a fast data path crossing the network with a single LSP. Since each LSP may or may not be associated with QoS parameters, such as link bandwidth, it is quite possible to have a huge number of LSP's on any given backbone link.

Another issue is user behavior: different users may put completely different requirements on reservations. For example, for IP telephony end-users, their reservations are likely to be short (measured in minutes), however, they demand fast call and reservation setup (measured in seconds). On the other hand, the inter-provider reservations can have very long duration (days or even months, depending on the business arrangement negotiated by the lawyers). Typically, fast reservation setup is not among the high priority requirements for the providers. Instead, once established, the reservations must always be available even when the network topology has changed.

In conclusion, we believe that signaling protocols must be able to carry control information for QoS reservations as well as for MPLS LSP setup. Managing large number reservation states in the network poses scaling problems.

4.2 The Hierarchical Reservation Model

As the Internet grows, both in size and in the diversity of service requirements, providing adequate reservation capability that can accommodate both of these factors becomes increasingly crucial. We propose a scalable hierarchical reservation model consisting of two major components: *inter-domain reservation* and *application-layer reservation*.

The inter-domain reservation component pre-computes and installs reservations that are shared by a significant number of end-users and networks at the routing domain level. Only service providers are authorized to establish and maintain these reservations. Inter-domain reservation signaling must rely on techniques such as aggregation and service abstraction to satisfy storage, bandwidth, processing and management constraints.

The application-layer component provides on-demand processing and installation of application-specific reservations that are initiated by the end-users. The potentially large number of different application-driven reservations, combined with the dynamic nature of these reservations (*i.e.*, frequent joins and departures), make them too costly to support at provider level. Thus, we need to rely on simple and lightweight signaling mechanisms to setup the reservations at the application level.

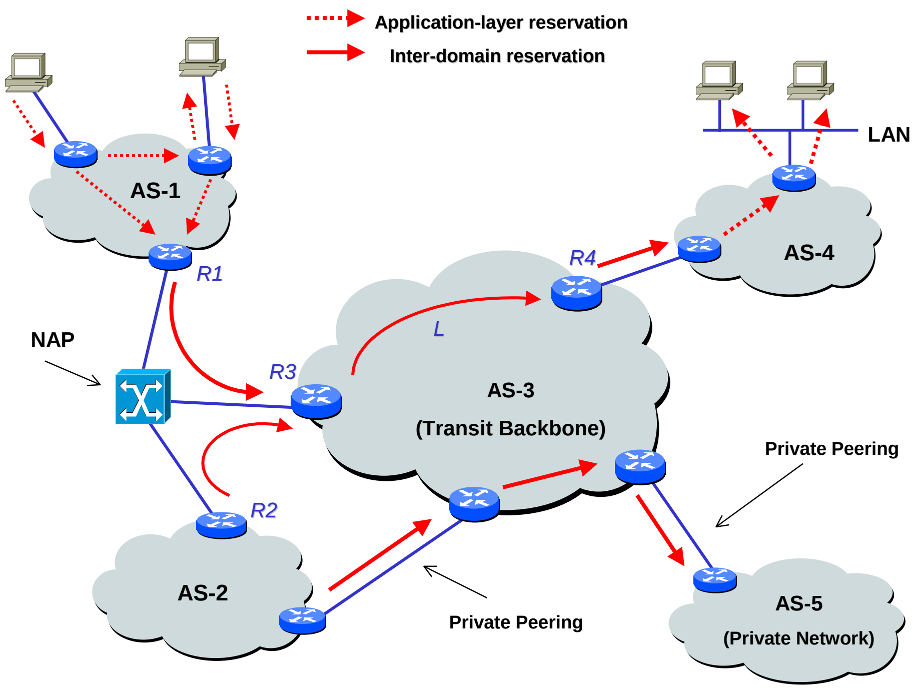


Figure 4-4: The Hierarchical Reservation Model

Figure 4-4 is an illustration of the hierarchical reservation model.

4.2.1 Inter-domain Reservation

From border routers, providers establish inter-domain reservations at NAPs, and POPs and on private peering links. The reservations can span multiple domains. Reservations are based on the bilateral (or multilateral) agreements between every *two* neighboring domains. There are many benefits in providing domain-level resource reservation. For security purposes, it hides parts of the actual provider's physical topology. Since inter-domain reservations combine each domain's resource information into a single entity, they can reduce the storage and maintenance overhead at routers. Another method to reduce the number of flows is by applying domain-level reservation aggregation. For example, in the figure, the border router R3 at AS-3 aggregates two incoming reservations from AS-1 and AS-2 going to AS-4. At AS-4, the routers need only to support a single aggregated reservation flow. The reservation details of AS-1 and AS-2 are thus hidden.

The ability to aggregate and subsequently abstract reservation requests is essential to the scaling of the model. This is especially true with respect to the inter-domain reservation component, since it must be capable of providing reservations to *all* or *almost all* reachable destinations. As recommended previously, aggregation based on either source or destination can keep the total number of reservations at a manageable level.

A natural choice for delegating a *domain* is to adapt the *routing domains* or *Autonomous Systems* as been defined and used in BGP. It is chosen for the following reasons:

- We need to preserve each routing domain's independence and autonomy, which is crucial for the deployment of resource reservation in the Internet.

- For good scaling, we need to have as little global coordination as possible. Each routing domain can operate relatively independently of each other [98].

- Resource reservation signaling relies on the underlying routing protocols to select paths. At the same time, reservation mechanisms need to be immune from the effects of route instability. Provided that BGP has been well engineered to deal with route looping, flapping and convergence, it is desirable to tightly interface the inter-domain reservation process with BGP routing.

As our inter-domain reservation component, we propose to create reserved "trunks" connecting multiple domains, but allow each domain to apply its own favorite method to manage internal transit traffic. We believe this is a feasible and reasonable approach.

Recent Internet traffic engineering studies [99, 100, 101] have been centered around the architectures and mechanisms to manage *intra-domain* traffic. Several providers, including UUnet, Level-3 and AT&T, have been in the process of deploying routers and tools to regulate internal traffic. However, there have not been much activity on how to bring these well-engineered provider networks together to achieve predictable *inter-domain* services.

Support for inter-domain reservation should be flexible. Specifically, the model should allow for a domain to participate in multiple reservations. For example, in Figure 4-4, AS-2 needs to setup up a reservation with AS-4, while it needs to transfer another group of user's traffic through a private peering arrangement to AS-5. AS-2 can initiate two reservations, to AS-4 and AS-5, from two different border routers. Both flows will go through the AS-3 network. While the example describes the case where a domain's traffic enters another domain from two different locations, there may be other cases where the traffic from one domain is sent to multiple neighboring domains at the same time (for traffic balancing purposes).

Providers need to install transit policies and route selection policies on border routers to regulate incoming traffic. To process these policies, border routers need to interface with BGP and traffic engineering modules. For example, in Figure 4-4, initially, a transit policy entry at router R3 is the following: *only allow packets from AS-1 to be transmitted to AS-4 with maximum guaranteed bandwidth B.* Upon receiving the policy entry, R3 interfaces with BGP and finds out that the exit border router to AS-4 is R4, and traffic from AS-1 must come from R1[4]. It then interfaces with the traffic engineering module to set up a reservation L with bandwidth B to R4. Finally, R3 updates its forwarding table to direct all the routes that are advertised from (AS-1, R1) to use reservation L.

In Chapter 7, we present a scalable inter-domain reservation protocol, BGRP (Border Gateway Reservation Protocol), that uses BGP to select the reservation path, and applies flow aggregation along a sink tree to reduce the total number of reservations.

[4]BGP ORIGIN, AS-PATH and NEXT-HOP attributes

4.2.2 Improving Scalability of Application-Layer Reservation

First, we need to put the necessity for using application-layer reservations in perspective. In all likelihood, only real-time streaming applications need to reserve network resources. Such applications include inter-active networking games, video teleconferencing and IP telephony. Within many well-engineered and well-provisioned networks, there is probably no need for using reservations at all. Our focus here is to provide reservation capability in networks that are occasionally congested.

By using an inter-domain reservation component, we have alleviated the pressure to improve scalability in application-layer reservation protocols, since a large percentage of inter-domain traffic will be aggregated and forwarded via the inter-domain reservation *trunks*. However, the use of inter-domain reservation does not address another major scaling problem associated with application-layer signaling: that of providing a large number of short-lived (in minutes) reservations in a timely fashion.

To improve the scaling property in application-layer reservations, we need to understand the criteria for fast reservation establishment, reservation processing capacity and the coordination between users and routers to recover from admission control failures. In the next chapter, we will describe these issues in detail. It is shown that signaling reservations in one pass from traffic sender's direction is the most efficient method. By using techniques such as reservation retry, and roll-back, we can recover from reservation failures quickly.

Then in Chapter 6, we illustrate an application-layer reservation protocol for real-time streams. The new protocol is very lightweight, allowing to create up to 10,000 reservations per second in routers.

Chapter 5

Designing Scalable Signaling Protocols

In this chapter, we discuss a number of design choices in reservation signaling that can effect the protocol scalability. To have a good grasp on the issue, we first need to understand: *why is implementing RSVP hard?*

Initially, RSVP was perceived as a lightweight protocol, in comparison, for example, with ATM signaling protocols such as Q.2931 [102]. However, as implementations are weighing in at over 30,000 lines of code [62, 63], it seems appropriate to review its design features that contribute to the complexity:

Receiver orientation: In RSVP, receivers make reservations, based on information provided by senders. This allows individual receivers within a single multicast group to request different levels of service guarantees, including none. This approach has the advantage of providing the flexibility to accommodate different reservation requirements from receivers in a heterogeneous multicast environment. However, this creates a great deal of processing complexity at routers.

For example, to support multicast, each RSVP session consists of one sender state, and multiple potentially distinct receiver states. RSVP needs to maintain each state individually via periodic refresh. The state management complexity only becomes worse when we deal with shared reservations with multicast, which requires multiple sender states mapping to multiple receiver states (*i.e.*, many-to-many mapping) for each RSVP session. In addition, we need to consider the dynamic nature of multicast group members. Since the implementation has to take into the considerations of frequent receiver joins and departures from a multicast group, the routers need to compute and merge reservations and propagate the changes to upstream routers in a timely fashion.

All of this causes excessive processing overhead, not to mention the need for large amount of storage memory, at routers. For many applications such as regular voice phone calls, receivers will simply request whatever traffic bandwidth the sender has indicated. Thus, it makes the need for receiver-orientated reservation questionable.

Two-pass reservation: Setting up an RSVP reservation takes at least two messages, PATH and RESV. PATH is responsible for interfacing with the underlying routing protocols to "pin-down" the reservation path, whereas RESV makes the actual reservation. This separation of path-finding and reservation messages imposes additional processing and protocol complexity. Experience has shown that state management is greatly simplified by requiring only one message (in either direction) to establish state, rather than going through several intermediate states.

Blockade state: RSVP uses *blockade states* to handle admission control failure. During RSVP RESV message processing, if a router cannot accommodate a reservation, it will notify the receiver about the failure. At the same time, the router stores a copy of the failed reservation request, and retries at the next refresh interval. To prevent these retries from blocking other reservation requests, the failed reservations must be managed separately from the rest of the reservation states. The failed reservation requests are referred to as *blockade states*. If a retry is successful, the router needs to notify the receivers by sending a confirmation message, and remove the reservation from the blockade state pool. As we will show later in this chapter, RSVP blockade states cannot recover from failure conditions quickly, and end-users may experience long reservation delays and frequent reservation rejections.

5.1 Design Choices

5.1.1 Reservation Model: One-pass vs. Two-pass

There are two basic approaches to reservation establishment in the network, one-pass and two-pass. A typical two-pass reservation model is the original Tenet architecture [103], in which the receiver sends a reservation request toward the source. Each network node along the way makes the reservation. Upon arriving at the source, the source sends another *relax* message back toward to the receiver, and has the option to modify the previous reservation at each node.

ST-II [94] is another two-pass reservation protocol. A sender originates a *Connect* message to a set of receivers. Each intermediate node determines the next hop subnets, and makes reservations on the links going to these next hops. Upon receiving a *Connect* indication, a receiver must send back either an *Accept* or a *Refuse* message to the sender. In the case of an *Accept*, the receiver may further reduce the resource request by updating the returned flow specifications.

In comparison, one-pass reservation protocols can make reservations at network nodes in a single pass. In case of a reservation failure, the failing nodes can always notify the reservation initiator. The one-pass approach has several attractive features. The most important one is that it simplifies the reservation process, since it does not go through several intermediate states to establish a reservation state at routers. However, the trade-off for using one-pass signaling is that the reservations are defined and initiated by one party (sender or receiver), and the other party may not have control over the level of reservation that it desires.

Originally RSVP was designed as a one-pass reservation protocol according to [73], where an end user (a receiver) delivers a reservation request sequentially to the admission control module in each network node along the data path, thereby making reservations along the way in a single pass. Unfortunately, since Internet routes are asymmetrical, the reservation receivers cannot possibly know the path that reserved data traverses. As a result, RSVP requests the senders to first initiate the reservation process by sending some messages (PATH) to pin down the forwarding data path before the actual reservation.

5.1.2 Path Pinning

Path pinning can only start from reservation sender's direction, since most routing protocols (practically all unicast routing protocols) are destination-based, and the Internet routing does not guarantee symmetrical data path, (*i.e.*, routing path from A to B is most likely different from the path from B to A). The main reason for such behavior is due to the "hot-potato" inter-domain routing policies installed at borders routers by the providers. The motivation here is to use the least network resources to carry other people's traffic. Therefore, there is no way for users to discover traffic forwarding routes from receiver's direction.

The reason for initiating reservations from receiver's direction (such as in RSVP) is to allow receiver diversity. But, at least for bandwidth diversity, reservations are an inappropriate means to distinguish classes of receivers. Bandwidth diversity could only be accomplished by "thinning" flows, i.e., dropping packets, as flows reach parts of the network endowed with less bandwidth. However, random packet dropping will quickly degrade most audio and video encodings due to their use of prediction across packet boundaries. Other mechanisms, such as layered multicast [104], were found to be superior to support diverse receiver populations. Unfortunately, in the absence of intelligent packet filters at the routers, receiver diversity is not likely to be useful.

Another argument for choosing receiver-based reservation is to support shared reservation, where multiple users create and use one reservation in the network. However, the important assumption of shared reservation is to have multiple data senders share one reservation on each link. This can be accomplished by either sender-based or receiver-based signaling protocols.

This leads us to conclude the following: *the only way to support one-pass reservation protocols in today's Internet is to combine path pinning and the reservation process, and allow senders initiate reservations.*

A possible one-pass reservation model could work as follows: Sender S initiates a reservation by sending a flowspec to all receivers D (assuming multicast). The message that carries the flowspec propagates through the network toward the receivers following the routing path. Each router along the way attempts to perform a resource reservation upon receiving the flowspec. If there is an admission error, the router caches the flowspecs for future reservation retries, tags a failure indication to the flowspec message and passes the flowspec to the next router.

This is the basic protocol design for YESSIR, a lightweight reservation protocol that we will further elaborate in chapter 6.

5.1.3 Partial Reservation and Reservation Blocking

We define partial reservations as follows: For a reserving flow, $f_{a \rightarrow b}$, let $\mathcal{L} = \{L_1, L_2, \ldots, L_n\}$ be the set of network links at the flow traverses, and $\mathcal{R} = \{R_1, R_2, \ldots, R_m\}, m \leq n$ be the set of the network links that have made reservations for the flow.

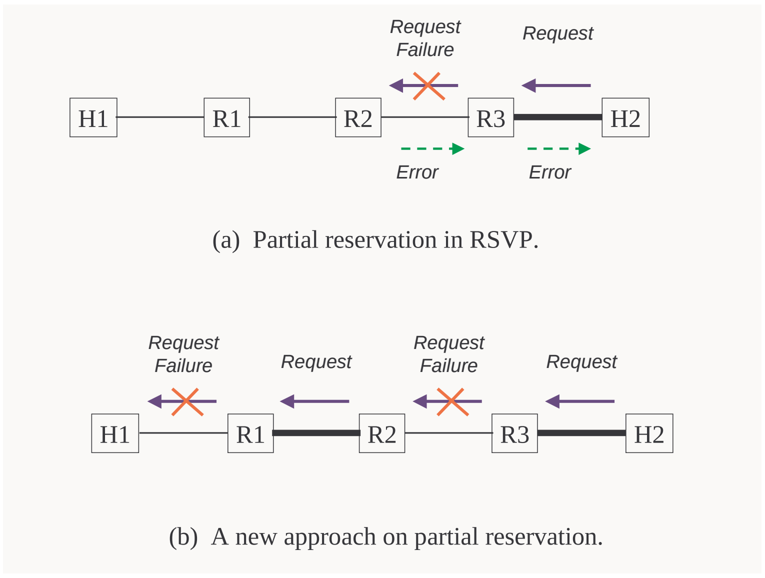
```
**if** $\mathcal{R} = \mathcal{L}$
    $f_{a \rightarrow b}$ is *fully* reserved.
**else**
    $f_{a \rightarrow b}$ is *partially* reserved.
```

RSVP can result in partial reservations: A reservation request that fails admission control creates blockade state and proceeds no further. The corresponding reservation is left in place in nodes downstream (towards the receivers) of the failure point. Upon receiving an admission control failure notification, the receivers detect that the reservation has been *blocked*, and have the option to either withdraw the previous request by sending an explicit tear-down message upstream, or sending a modified reservation request.

Generally, the reason for a reservation failure is the lack of sufficient resources, i.e., bandwidth or buffer space, to accommodate the reservation at the time of the request. Reservations can also fail because of route flapping, packet loss or router software failure. However, these failures are not relevant in the context of our discussion here.

Here we need to be careful with our assumption on the occurrence of resource shortage and subsequent reservation blocking. As emphasized in Chapter 2, resource reservation is not a tool to compensate for resource inadequacies caused by over-subscription, and sloppy network design. Applying resource reservation in an over-subscribed network can only cause service degradation for everyone, and aggravating congestion conditions. Rather, resource reservations are only useful for protecting flows on links that are congested occasionally and for short time periods. Hence, we argue that in a reasonably well-managed network with a large number of data flows, reservations start and terminate at a high rate, causing the resource shortage likely to be a *temporary* one. This suggests keeping the partial reservations instead of retrying the reservation from scratch at a later time.

The same rationale argues against the way that RSVP is handling partial reservations via blockade states. Just because a reservation has failed on one link along a reservation path does not mean the rest of the links will fail, too. A preferred reservation approach would be that *if a reservation request is denied, the reservation request still advances to the next hop*. For the same network, it will create more reserved links than RSVP does.



(a) Partial reservation in RSVP.

(b) A new approach on partial reservation.

Figure 5-1: Partial reservation comparison. The bold lines represents the links that can successfully make a reservation.

Figure 5-1 illustrates a comparison between RSVP and the new approach. With the new approach, each request tries to make reservations on as many links on its path as possible. It helps obtaining more resources for the requesting flow and potentially speeds convergence to a fully-reserved path. As shown in the figure, there is an admission failure on link R2-R3. In RSVP, a blockade state is created at R2, and the reservation process is terminated. In the new approach, the reservation request is forwarded to the next router regardless of admission control failures, as a result, more links (R1-R2 and R3-H2) have completed reservations despite temporary resource shortage at both H1 and R2. Section 5.3.2 will show simulation results to further illustrate this point.

5.2 Fast Reservation Setup

5.2.1 Reservation Retry and Resource Grabbing

Partial reservations do not provide the service quality that end users have originally requested. Hence, it is desirable for routers to "fill in" the missing reservations as soon as possible, since the tolerance for session set up delay is limited to a few seconds for many applications. We call the process of attempting to complete the reservations along the path *reservation retry*.

RSVP provides one form on reservation retry, where it combines the retry with soft-state refresh. Since the routers periodically send flow states to the neighbors, the routers can retry the reservations at each refresh cycle. However, a refresh cycle can be quite long. For example, the default refresh timer is 30 seconds for RSVP. This may be too long in applications such as Internet telephony where human users are waiting for the results and session life times are only a few minutes. Also, in Chapter 3 and [85], there have been recommendations to dynamically adjust the refresh frequency based on network conditions to improve the reliability in soft-state messaging. Hence, retrying failed reservations only at soft-state refresh intervals may not be good enough.

We here introduce a more aggressive method for partial reservation retries. We define a reservation pending queue, $\mathcal{Q}$, which is structured as a FIFO (First-In-First-Out) chain:

$\mathcal{Q}$.head	pointing to the first arriving entry
$\mathcal{Q}$.tail	pointing to the last arriving entry

For each reservation flow, f, it has the following attributes:

f.resource	the amount of desired resource
f.prev	points to the previous flow ahead in the queue
f.next	points to the next flow

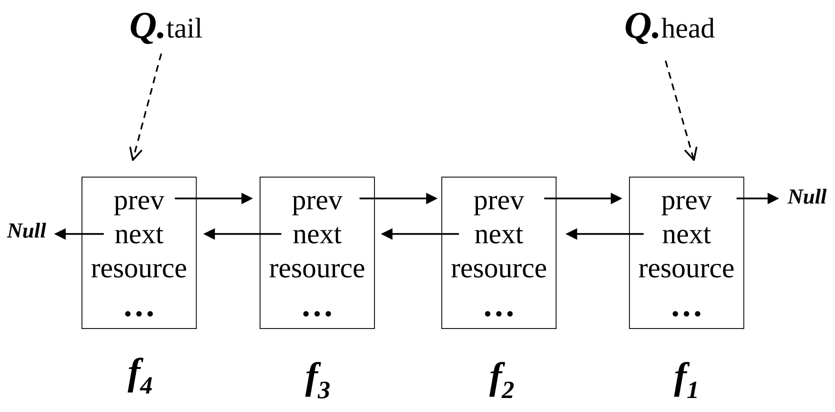


Figure 5-2: Reservation Pending Queue Structure

Figure 5-2 shows the relationship between $\mathcal{Q}$ and reservation flows.

Assume that a flow f_i needs to reserve f_i.resource resources, while the reservable link resource is $\mathcal{B}$. There exists another flow f_j with reservation f_j.resource on the link. The algorithm is shown in Figure 5-3.

At reservation time, if there are not enough resources available, the request is queued in $\mathcal{Q}$. As soon as extra resource become available (in this case, another flow f_j has terminated), the router dequeues the request from $\mathcal{Q}$ and retries failed reservations. We refer to this scheme as *resource grabbing* and will demonstrate its effectiveness in Section 5.3.2.

Note that partial reservations and fragmentation (see below) are only likely to occur

```
1: on receiving the reservation request for $f_i$:
        **if** $f_i$.resource $\leq \mathcal{B}$
                $\triangleright$ update the reservation;
                $\mathcal{B} \leftarrow \mathcal{B} - f_i$.resource;
                reserve $f_i$.resource on the link;
        **else**
                $\triangleright$ queue the request;
                enqueue $f_i$ to $\mathcal{Q}$;
        **return**

2: when $f_j$ is deleted or timed out:
        $\mathcal{B} \leftarrow \mathcal{B}$ + $f_j$.resource
        $\triangleright$ search through the queue
        **if** $\exists$ flow $f_x$ in $\mathcal{Q}$ such that $f_x$.resource $\leq \mathcal{B}$
                dequeue $f_x$ from $\mathcal{Q}$;
                $\mathcal{B} \leftarrow \mathcal{B}$ - $f_x$.resource;
                reserve $f_x$.resource on the link;
        **return**

3: $<$enqueue flow $f$ to a FIFO queue $\mathcal{Q}$ $>$:
        **if** $\mathcal{Q}$.tail = NULL
                $\mathcal{Q}$.head $\leftarrow f$;
                $\mathcal{Q}$.tail $\leftarrow f$;
                $f$.prev $\leftarrow$ NULL;
                $f$.next $\leftarrow$ NULL;
        **else**
                $\mathcal{Q}$.tail.back $\leftarrow f$;
                $f$.prev $\leftarrow \mathcal{Q}$.tail;
                $\mathcal{Q}$.tail $\leftarrow f$;
                $f$.next $\leftarrow$ NULL;

4: $<$dequeue flow $f$ from a FIFO queue $\mathcal{Q}$ $>$:
        **if** $f$.prev = NULL
                $\mathcal{Q}$.head $\leftarrow f$.next;
        **if** $f$.next = NULL
                $\mathcal{Q}$.tail $\leftarrow f$.prev;
        **else**
                $f$.prev.next $= f$.next;
                $f$.next.prev $= f$.prev;
```

Figure 5-3: Reservation retry algorithm and supplemental procedures

in heavily loaded networks or under overload conditions. However, this is exactly when reservation is needed at all – best effort service works fine in an under-utilized network.

5.2.2 Fragmentation Recovery

Partial reservation can lead to reservation fragmentation, where a large number of flows all have partial QoS, but all with unacceptable quality. An analogy to this is the deadlock problem in operating systems, where multiple processes try to access the same set of resources, and are all waiting for others to release theirs first. Since no process is willing to release the resource, a deadlock occurs. In Section 5.3.3, we will demonstrate the effect of reservation fragmentation.

Note, however, that the analogy is not complete. An operating system task can make no progress as long as it is missing one resource, while a data flow may decide to "risk" the QoS degradation at a small number of routers, in the hope that the admission control mechanism is conservative and that there is enough best-effort bandwidth available.

There are a number of studies [105, 106, 107, 108] over the years on deadlock recovery. To grasp a better understanding of the problem, we propose the following over-simplified methods for partial reservation recovery:

Preemption: A flow with high priority can take resources away from lower-priority flows holding these resources.

Rollback: All flows withdraw their partial reservations, and re-request at some random time later.

Suspend hopeless flows: Flows that have consecutively failed their end-to-end reservation attempts too many times are simply ignored by routers, leaving resources for other flows.

The first two solutions require cooperation from end users, and thus more messaging between routers and end users. Plus, they do not prevent "impolite" users from persistently asking for reservations and obtaining as many network resources as they can. For example, one method to enable preemption is to have end users monitoring reservation conditions inside the network. If there is a failure, the end user has the option to increase the priority level of the flow, and re-initiates the reservation process. We plan to investigate end-user reservation preemption and rollback in the future.

In the rest of the chapter, we study the performance of the third option, where routers suspend the reservation retries on a flow if it has failed too many times. Figure 5-4 presents the algorithm that allows routers to limit the number of retries. It is based on the *resource grabbing* algorithm in Figure 5-3. We define a threshold, $\mathcal{T}$, as the maximum number of retries that a flow can exercise during its entire duration at a single router. Each reservation flow, f, needs to maintain a retry count f.count in addition to the reservation amount f.resource.

```
1: on receiving the reservation request for $f_i$:
        **if** $f_i$.resource $\leq \mathcal{B}$
                $\triangleright$ update the reservation;
                $\mathcal{B} \leftarrow \mathcal{B} - f_i$.resource;
                reserve $f_i$.resource on the link;
        **else**
                $\triangleright$ queue the request;
                enqueue $f_i$ to $\mathcal{Q}$;
                $f_i$.**count** $\leftarrow f_i$.**count** $+ 1$;
        **return**

2: when $f_j$ is deleted or timed out:
        $\mathcal{B} \leftarrow \mathcal{B}$ + $f_j$.resource
        $\triangleright$ search through the queue
        **if** $\exists$ flow $f_x$ in $\mathcal{Q}$ such that $f_x$.resource $\leq \mathcal{B}$ and $f_x$.**count** $\leq \mathcal{T}$
                dequeue $f_x$ from $\mathcal{Q}$;
                $\mathcal{B} \leftarrow \mathcal{B}$ - $f_x$.resource;
                reserve $f_x$.resource on the link;
        **return**
```

Figure 5-4: Modified reservation retry algorithm that uses a threshold to control fragmentation recovery.

By adjusting the value of $\mathcal{T}$ on routers during congestion, the fragmentation effect can be reduced. Here, we only limit the retry attempts during the resource grabbing process; flows can continue to retry failed reservations every refresh cycle.

5.3 Simulation

5.3.1 Simulation Methodology

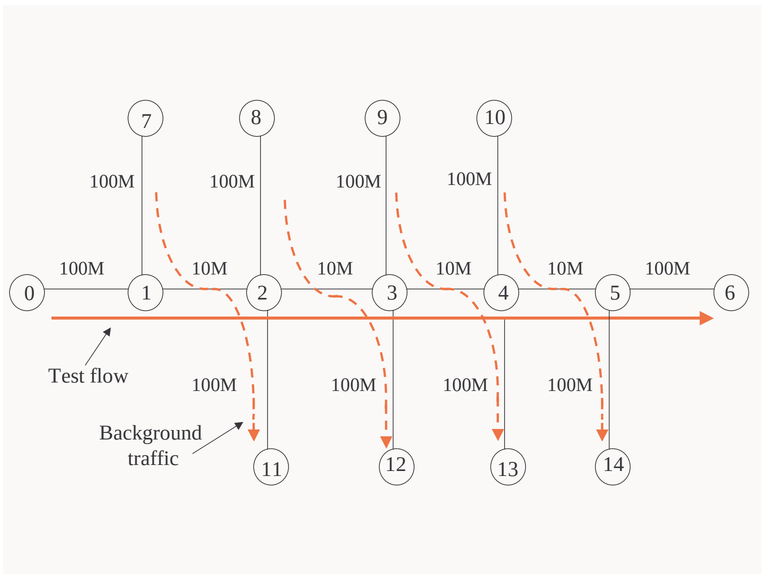


Figure 5-5: The network topology used in the reservation retry simulation.

To evaluate the design ideas described in Section 5.1, we used the *ns* simulator and its RSVP module, and extended it to support partial reservation functionality required for our experiments.

Figure 5-5 shows a 15-node simulation network topology. Nodes 1, 2, 3, 4 and 5 are backbone nodes, and the remainder are end systems. All backbone links have 10 Mb/s bandwidth and 20 ms propagation delay; all access network links have 100 Mb/s bandwidth and a propagation delay of 10 ms. Network links are reliable. We assume that each link has a high-priority queue reserved for reservation messages so that they are never lost. Up to 50% of the link bandwidth is reservable, and the rest of the bandwidth is shared by best-effort and reservation traffic.

In the simulations that follow, network nodes 7, 8, 9 and 10 generate best-effort data flows as well as reserved data flows to nodes 11, 12, 13 and 14, respectively. All data packets are 125 bytes long. The best-effort data flows are modeled as exponential on/off traffic source, with on-time 1 s, off-time 0.5 s, a burst rate of 500 kb/s and an exponentially distributed flow duration with a mean of 150 s. The reserved flows are all CBR traffic with rate r of 100 kb/s and a token bucket size B of 5,000 bytes. The reason for choosing CBR is to simulate the reservations used for voice streams. We can create various network congestion conditions by adjusting the number of best-effort and reservation flows.

We assess the effectiveness of different reservation algorithms by monitoring a CBR flow from node 0 to node 6 traversing several congested links. In each simulation experiment, node 0 starts transmitting a 100 kb/s flow at time 0 to node 6. 250 s later, node 0 then tries to reserve resources for the flow. The reservation session lasts 300 s. We monitor this test flow at node 6 by capturing the data rate received. When the end-to-end reservation has been completed, we see a fixed rate of 100 kb/s (that is, a flat line on the traffic trace diagrams).

Data for the first 50 seconds in each simulation are discarded to obtain steady-state results. Each simulation has been run several times with different random seeds.

5.3.2 Partial Reservation

In a first experiment, labeled "regular load" in the figures, we created a mildly congested network with 27 best-effort flows and 85 reservation sessions in the background. The background best-effort traffic is about 1.35 times more than the link capacity at each hop. Since the total number of reservation sessions exceeds what the backbone routers can handle, we expect to see many reservation rejections, where rejected flows wait and retry.

All reservation protocols tested are soft-state based with a 30-second average refresh interval. To avoid synchronization, refresh intervals are randomly varied between 21 and 39 s, as recommended in RSVP.

Figure 5-6 shows the packet rate received at node 6 in a 600 s simulation. All nodes in

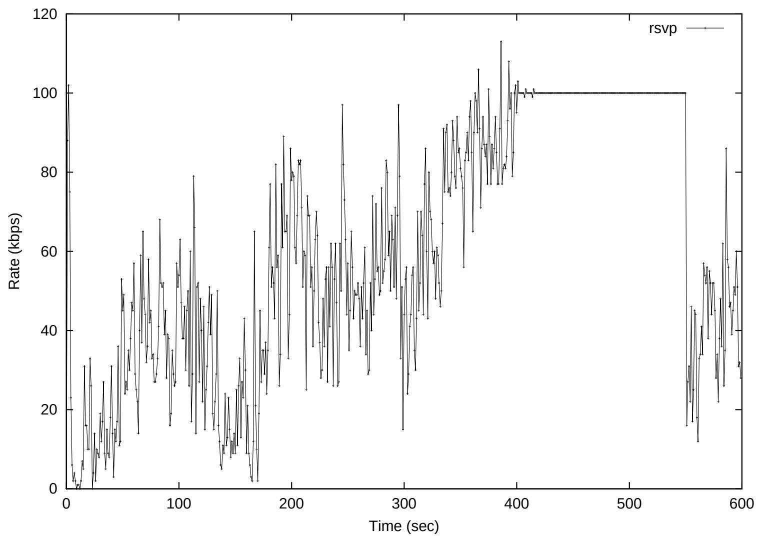


Figure 5-6: Regular load: RSVP reservation: reservation completed after 150 s and 5 tries.

the network use RSVP for reservation. A rejected flow can only retry for the reservation at the next refresh cycle. The test flow takes about 150 seconds and five tries to complete the reservation.

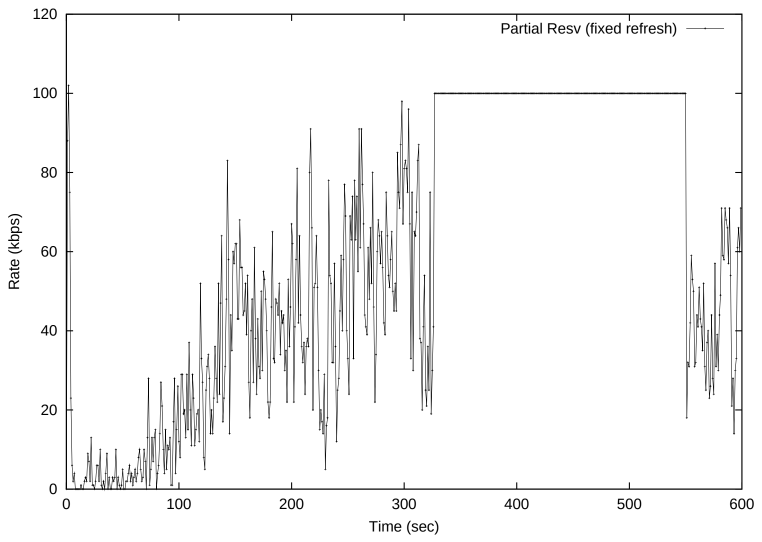


Figure 5-7: Regular load: one-pass reservation without grabbing; reservation completed after 77 s and 3 tries.

We then ran the same identical testing scenario with an one-pass reservation mechanism that uses soft-state refresh to retry the failed reservations. As shown in Figure 5-7, the reservation completes after 77 seconds and 3 tries.

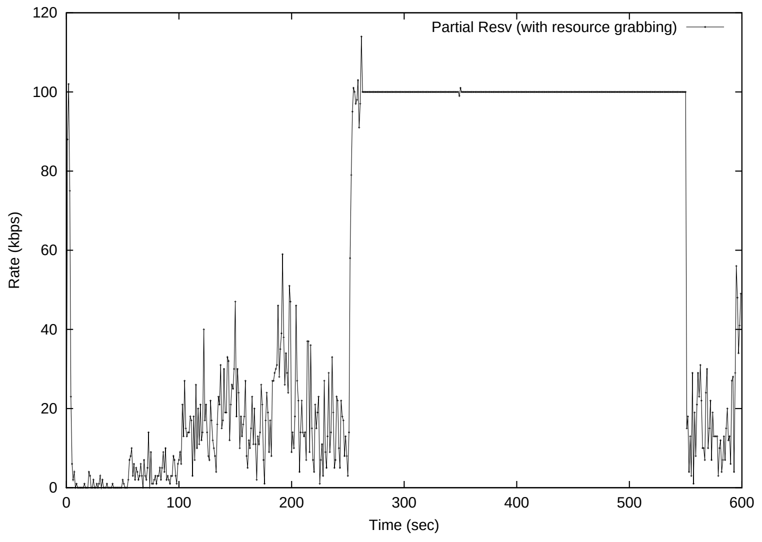


Figure 5-8: Regular load: one-pass reservation with resource grabbing; reservation completed after 12 s and 19 tries.

Figure 5-8 shows a scenario where all the nodes use one-pass soft-state reservation and, in particular, the resource grabbing mechanism that we have described in Section 5.2.1. The testing flow reservation takes only 12 seconds to complete, but retries 19 times.

We then collected reservation failure and success data from all the nodes. Figure 5-9 shows how the flow had completed the reservation in the three testing scenarios above.

Using RSVP, the flow received reservations from node 5 and 6 during its first reservation attempt, but was rejected at node 4. At the next refresh cycle, the flow made the reservation on node 4, but was rejected from its immediate upstream node, node 3. It took two refresh cycles for the flow to eventually get a reservation from node 3. During the fifth refresh cycle, the flow made reservation on the link from node 2 and 1, and thus completed the reservation.

Even though the RSVP flows do not have to start reservations from scratch after each reject, due to blockade states, it takes a long time for the flow to make its way toward the sender, often one hop per refresh interval. In comparison, the one-pass reservation scheme can perform much better. At the reservation initiation time, the request message passes

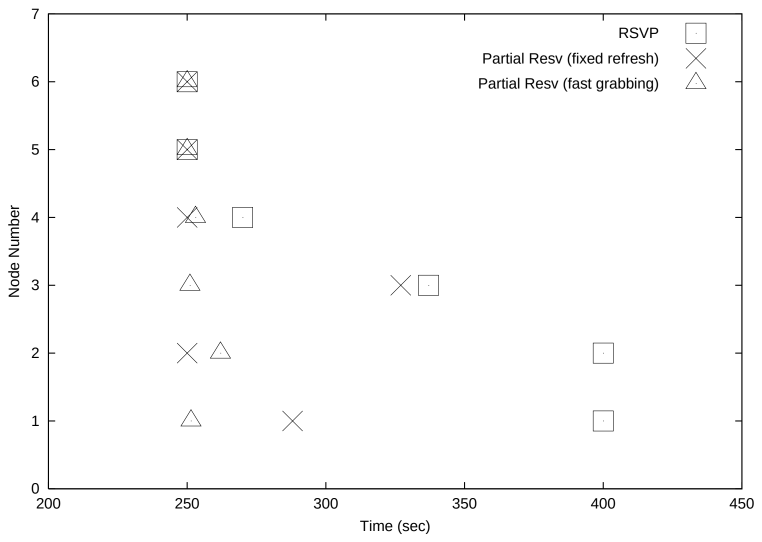


Figure 5-9: Regular load: Reservation sequence for RSVP, one-pass reservation with refresh, and one-pass reservation with refresh plus resource grabbing. The ordinate shows the node number in the simulation network of Figure 5-5. The figure shows the amount of time it would take to setup a successful reservation.

through all the nodes on the reservation path, and tries to make a reservation at each node. As shown in the figure, the flow made the reservations on nodes 2, 4, 5 and 6 during its first reservation attempt. It took two more refresh cycles to complete the reservation.

In conclusion, the reservation scheme armed with resource grabbing mechanism performed the best. It completed the reservation in 12 seconds, and retried reservations 19 times in total on all 6 nodes during this period.

5.3.3 Fragmentation Recovery Performance

To evaluate the reservation retry performance, we increased the number of background reservation flows to 120, that is to request 2.4 times more resource than what the network can provide. We experiment with both RSVP and the one-pass reservation scheme in this highly congested network. We also apply the threshold algorithm defined in Figure 5-4 on all network nodes when run the one-pass reservation experiments.

Table 5.1 collects the total number of reservation retries from all network nodes, and the statistics on setting up an end-to-end reservation from node 1 to 6.

Description	# of retries on all nodes	Test flow between Node-0 and Node-6
One-pass Reservation, $\mathcal{T} = \infty$	28,314	no reservation made in 184 tries
One-pass Reservation, $\mathcal{T} = 10$	8,534	no reservation made in 46 tries
One-pass Reservation, $\mathcal{T} = 8$	7,137	reservation completed after 232 s and 32 tries.
One-pass Reservation, $\mathcal{T} = 3$	4,382	reservation completed after 172 s and 14 tries
One-pass Reservation, $\mathcal{T} = 1$	2,685	reservation completed after 19 s and 1 retry
One-pass Reservation, $\mathcal{T} = 0$	2,588	reservation completed after 43 s and 3 tries
RSVP, fixed refresh	3,409[1]	no reservation made in 10 tries

Table 5.1: Number of retries in the network, measured for a 550 s simulation duration.

From simulation, we observed that RSVP could not make the end-to-end reservation. This is probably due to its ineffectiveness during reservation retry. On the other hand, the one-pass reservation with very aggressive retry process (high $\mathcal{T}$) failed to make end-to-end reservation as well. An explanation is that, with an aggressive retry algorithm, all reservation flows in the network try to take as many resources as possible. As a result, few flows get the full reservation.

With a proper threshold value, the user flow can successfully make the reservation in a highly congested network. In the simulation, the best scenario is the one with $\mathcal{T} = 1$. We suspect that with lower threshold number, the flows are less aggressive to retry for the link resource, and therefore allow other flows to complete their reservations. This conclusion seems to be supported by tests using different $\mathcal{T}$. We also conclude that in using the one-pass reservation with retry, there is a trade-off between the ability to obtain resources quickly, and the likelihood of causing reservation deadlocks.

RSVP is also less aggressive in grabbing resource with its blockade state algorithm, then why does it perform so poorly? From our collected data, out of 3,409 reservation retries, there were only 739 successful flows in the 550-second simulation interval. Since the blockade states make partial reservations only on the nodes that are downstream from the failure node, RSVP flows thus receive less resources from the network, and thus perform poorly.

[1]We have also monitored the total number of new RESV messages being received at the end nodes, which is 739. This is the same as the total number of successful RSVP reservations.

5.4 Discussion

We have investigated two important design issues in a reservation signaling protocol design: reservation retry and one-pass reservation.

Through simulation, we have compared the reservation performance achieved from RSVP and an one-pass reservation scheme. RSVP uses a more conservative hop-by-hop reservation retry mechanism, thus it may take a long time to achieve reservation convergence in resource-constrained networks. On the other hand, a better alternative is to retry reservations on multiple links in parallel. We also discovered that routers need to control the reservation retry process of the failed flows. One such mechanism is to limit the number of reservation retries on failed flows.

In dealing with partial reservation recovery, we studied a method of suspending failed flows. Though the study has shown some promising simulation results, a far more interesting work will be to understand how to avoid reservation deadlock conditions in the network.

Chapter 6

YESSIR: An Application-Layer Reservation Protocol

6.1 Introduction

In this chapter, we incorporate the protocol design ideas studied in Chapter 5 to develop a lightweight resource reservation protocol, YESSIR (YEt another Sender Session Internet Reservation[1]). It is an in-band, sender-based, one-pass protocol based on RTP that offers significantly lower code and run-time complexity than RSVP.

YESSIR is motivated by the observation that a large fraction of the applications that require guaranteed quality-of-service are continuous media applications and that a substantial fraction of these either use or will use the Real-Time Transport protocol (RTP) [24] to deliver their data. YESSIR and RSVP can operate side-by-side in the same network, without affecting the certainty of guarantees offered to applications.

In the remainder of this chapter we describe several relevant protocols (section 6.2), enumerate the specific design objectives for light-weight reservation signaling (section 6.3), and describe how YESSIR is designed to realize these requirements (section 6.4 and 6.5). In Section 6.6, we present the implementation results.

[1]The name reflects the proper attitude of a resource reservation protocol in a well-designed network.

6.2 Relevant Protocols and Features

6.2.1 RTP

RTP has been designed to provide end-to-end delivery services for data with real-time characteristics. Although protocol-independent, applications normally run RTP on top of UDP to make the use of its multiplexing and checksum services. It has been widely implemented on multimedia systems across all operating systems and is part of the ITU H.323 recommendation [109] for conferencing and Internet telephony. Examples include *vic* [110], *vat* [111], *rat* [112] and *NeVoT* [113] for teleconferencing over the MBONE, *NetMeeting* from Microsoft and conferencing tools from Netscape.

Although RTP was not intended as a resource reservation protocol, resource reservation can benefit from the following RTP features:

In-band signaling: RTP defines two types of packets: RTP for data transport and RTCP for control. Each RTP session consists of one RTP data stream and one corresponding RTCP stream, originated by one or more participants. When carried over UDP, data and control packets use adjacent port numbers, so that a router or firewall can easily map from a control stream to the corresponding data stream.

Periodic sender/receiver notification: Senders and receivers periodically send RTCP packets containing reports to the multicast group. Data senders distribute sender reports (SR's) that indicate, inter alia, the number of bytes and packets transmitted since the last report and information allowing the estimation of round-trip times. Data receivers include receiver reports (RR's) that indicate packet loss and delay statistics, among others. By evaluating these reports, all participating members have knowledge of traffic characteristics, network congestion and session membership. Routers can deduce the resource requirements of a session from these reports, as will be discussed below.

The period between reports has a minimum of five seconds and scales with the number of participants, keeping the RTCP session control overhead limited to no more than 5% of the data bandwidth. Senders are allocated at least 25% of the session control bandwidth.

Embedded in applications: RTP is typically implemented as part of the application. As will be shown in Section 6.4.1, even an RTP application that runs the current version of RTCP can be used to initiate resource requests. No kernel modifications beyond the support of IP router alert options are needed.

6.2.2 IP Router Alert Option

The IP router alert option [88, 89] alerts transit routers to more closely examine the contents of an IP packet. In other words, routers can intercept packets not addressed to them directly, with little performance impact. For example, RSVP PATH messages are carried in IP packets that include the router alert option. Thus, even though RSVP PATH messages are addressed to end systems, PATH messages are intercepted and processed by all transit routers. We make use of router alert options to mark RTCP sender report for YESSIR processing.

Since existing UNIX kernels cannot easily deliver IP packets containing IP options to applications, we have defined and implemented a new protocol family called `PF_IPOPTION`for the FreeBSD kernel. More details are described in [90].

6.3 Design Objectives

YESSIR is designed to offer an alternative approach to resource reservation in the Internet, using RTCP sender reports to reserve resources in the network. We have several design objectives in mind when designing this protocol:

One-pass resource reservation: As described in Chapter 5, the one-pass reservation model is more efficient than a two-pass model. Due to the constraints in IP routing, the only way to support one-pass reservation is to have the sender initiates reservations. In addition, we argue that many applications cannot make full use of the benefits of receiver-initiated reservations. Sender-initiated reservation is also a better fit for policy and billing, as the number of senders making reservations is likely to be much smaller than the number of receivers in multicast scenarios. In many existing systems, such as cable television, the cost of “resource reservation” is bundled with the cost of content, simplifying billing. (Also, a provider of pay-per-view services would likely want to avoid the case where subscribers pay, fail to reserve resources and then ask for their money back since the quality was unacceptable.) In the absence of an Internet-wide authentication and cross-provider billing service, it is far easier for the relatively small number of large-scale content providers, residing at known network addresses, to arrange for payment with major backbone providers than individual subscribers.

Allow partial reservations: The function of resource reservations is to protect existing streams against disruption by other streams that arrive later.

In “classical” reservation systems, reservations are either made or denied end-to-end. Depending on the system, the requestor can always either ask again, at some cost to the network if done too often (“redialing”). Some systems also allow to specify a range of resource requests to increase the likelihood of success, however, this can cause low-bandwidth regions to experience high packet losses despite reservations.

As motivated in Chapter 5, partial reservation has the advantage of expediting the reservation process by obtaining resources on as many links as possible for a particular flow. On links without reservation, traffic is carried on a best-effort basis and the resource reservation request continues downstream towards the receivers. Applying different resource grabbing techniques, a flow can acquire a reservation on a link when another flow terminates, without having to retry at the application layer. The user can decide whether to put up with a partially successful reservation and hope that more links will be added as the session continues or cancel the session. For a live presentation, where inserting an end-to-end reservation means missing the event, a user may well decide that the prospect of improving reservation fortunes may be better than not listening at all or foregoing all resource reservations.

Provide different reservation styles: Reservation protocols should support both *individual* and *shared* reservation styles. Individual reservations are made separately for each sender, whereas shared reservations allocate resources that can be used by all senders in an RTP session.

Individual reservations are called for when all senders are active simultaneously, *e.g.*, for distribution of participant video in a conference, while shared reservations are appropriate where several senders alternate, e.g., for audio in a conference. (Shared reservations also avoid the problem that a new speaker may not be able to acquire a reservation; they can re-use the existing reservation of the previous speaker.)

We need to design simplified versions of the fixed filter and wildcard filter reservations in RSVP. Note that the shared reservation styles, one of the distinguishing features of RSVP, does not depend on receiver orientation. We can handle the shared reservation style from the sender's direction, while RSVP supports shared reservation (shared-explicit and wildcard-filter styles) from the receiver's direction.

Robustness and soft-state: Similar to RSVP and PIM [91], routers maintain reservation states as *soft state*, *i.e.*, reservations disappear by themselves if not refreshed periodically. This avoids orphan reservations and can be adaptive to routing changes. As in RSVP, we can design an explicit teardown mechanism using RTCP BYE packets to avoid holding reservations for a number of soft-state refresh intervals after the requesting application has terminated.

In-band signaling: Rather than defining another signaling protocol, reservation messages can be transported by RTCP. Given that RTP's data and control packets are tightly coupled, updating packet classifiers and firewall filters can be greatly simplified.

Interoperability: Reservation messages are piggybacked in RTCP, but the operation of existing RTP functions at end systems is not affected at all. Reservation protocols need to interface with router's traffic control module to accomplish QoS functionality, but it cannot be constrained to any specific QoS technique.

Provide link resource advertising function: The purpose of making link-level resource reservation is to meet end-to-end application requirements. To that end, YESSIR needs to be able to query and carry collected network resource information to the end systems.

6.4 YESSIR Protocol

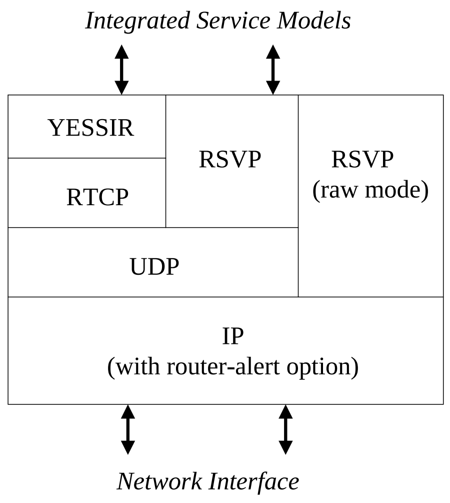


Figure 6-1: Protocol relationships.

YESSIR reservation messages consist of RTCP sender-report messages, possibly enhanced by additional YESSIR-specific data, carried in IP packets tagged with router-alert options. The placement and relationships to other protocols are shown in Figure 6-1.

Reservation requests generated by senders are intercepted and processed by those routers that support the router-alert IP option. Routers that do not support the option or YESSIR forward the RTCP message unaltered to the next hop. End systems ignore the router alert option. Thus, YESSIR can be deployed incrementally and without affecting the behavior of end systems.

6.4.1 Outline of Operation

The YESSIR protocol works as the following: Senders periodically multicast RTCP sender reports (SR's) to all members of the multicast group (or the other party, if unicast). The sender reports contain transmission and reception statistics and timing information. Reservation requestors may insert explicit reservation information into SR's, however, YESSIR can also operate without any additional information beyond what is already contained in RTCP sender reports (see Section 6.4.2). When an RTCP SR is received by a router, the router will attempt to make a resource reservation according to the information specified in the message.

If a reservation request cannot be granted at a router, the RTCP SR packet will continue to be forwarded to the next hops. The router has the option of inserting reservation failure information into the SR. As a part of RTCP receiver reports (RR), the receivers will provide failure information to the senders. Based on RRs received, senders can either drop the session, or lower the reservation request and transmitted bandwidth.

If a reservation request is accepted by a router, the corresponding RTP data stream information is added into the packet classifier, and the router's scheduler is updated to support the new stream.

Instead of basing reservations on explicit flowspecs, YESSIR can operate in a *measurement-based* mode. One type of measurement-based reservation is to make use of the fact that RTCP SR's contain a byte count and a timestamp. If the first RTCP packet for a session does not contain a flowspec, the router simply records the timestamp and byte count, but does not make a reservation. If a second packet for the same session comes along, the router computes the difference in time stamps and byte counts and thus computes an estimated rate. It then establishes reservations for this measured bandwidth, updated as new RTCP packets arrive. Compared to other measurement-based admission controls [114, 115, 116], this frees the router from the burden to count packets and estimate rates. We will describe measurement-based reservation below in more detail.

Reservation states in each router are maintained as soft-state. The reservation is automatically removed if no RTCP SR is received within several consecutive refresh intervals.

To reduce the processing burden at routers, instead of having routers to initiate refresh messages, RTP senders periodically generate RTCP SR's with IP router-alert option to refresh reservations. Compared with the hop-by-hop refresh mechanism that has been used in RSVP, the end-to-end refresh mechanism in YESSIR reduces routers' processing cost (as shown in Section 6.6.4).

In addition, an RTCP BYE message, sent when one or more traffic sources are no longer active, releases the YESSIR state record and any resource reservations.

6.4.2 Flow Specification

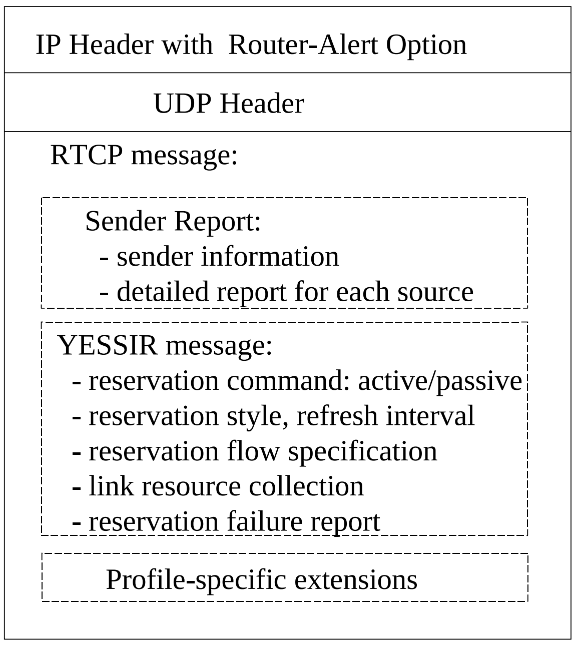

IP Header with Router-Alert Option
UDP Header
RTCP message:
Sender Report:
- sender information
- detailed report for each source
YESSIR message:
- reservation command: active/passive
- reservation style, refresh interval
- reservation flow specification
- link resource collection
- reservation failure report
Profile-specific extensions

Figure 6-2: YESSIR Message Structure with Explicit Flowspec.

YESSIR operates in two modes: *explicit* and *measurement-based*. Explicit mode requires the users to piggyback explicit reservation information in RTCP reports (SR or RR). We have defined an optional reservation extension for RTCP, as shown in Figure 6-2. The YESSIR extension consists of reservation style, refresh interval, and the necessary information as to whether to admit the flow and what resources to set aside. In addition, we defined a network monitoring fragment in the YESSIR extension. If it is present in a request, every router along the path needs to insert the network resource information of the reserved link in the message. This component is equivalent of the RSVP ADSPEC object [31] for the purpose of supporting "One Pass With Advertising" (OPWA) [73]. Routers where reservation requests fail indicate the reason for failure in the reservation error fragment. The fragment is used to collect error information that will allow end systems and network administrators to diagnose reservation failure inside the network.

In measurement-based mode, routers gather flow specification from the data packets directly, and thus free end-users from articulating reservation requirement details. There are several methods to gather reservation data:

- RTCP SR (Sender-Report Message),
- RTP PT (Payload-Type field), and
- IP Header DSCP (DiffServ Code Point).

The first method estimates the average user data rate by reading the timestamp and the byte-count in RTCP SR's. A standard RTCP SR message format is shown in Figure 6-3. The *NTP timestamp* uses the timestamp format of the Network Time Protocol (NTP), which is in seconds relative to zero hour UTC (Coordinated Universal Time) on 1 January 1900 [117]. It indicates the wallclock time when a RTCP SR was sent. The *sender's octet count* is the total number of payload bytes transmitted in RTP data packets by the sender since starting transmission up until the time this SR packet was generated.

The advantage in this method is that senders do not have to do anything other than periodically send regular RTCP SR's with IP Router Alert option, and routers can adjust reservation levels dynamically. However, it has several problems. First, the actual reservation does not take place until the arrival of the second RTCP SR. For the duration of first two RTCP SR's, which normally lasts 5 seconds, there won't be any reservation for user flows. Secondly, the estimated data rate represents the resource usage in the duration of two *previous* RTCP SR's. Hence, for bursty streams, this method may result in either under-reserving or over-reserving network resources.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|    RC   |   PT=SR=200   |             length            | header
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         SSRC of sender                        |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|              NTP timestamp, most significant word             | sender
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ info
|             NTP timestamp, least significant word             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         RTP timestamp                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     sender's packet count                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      sender's octet count                     |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                 SSRC_1 (SSRC of first source)                 | report
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ block
| fraction lost |       cumulative number of packets lost       |   1
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           extended highest sequence number received           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      interarrival jitter                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         last SR (LSR)                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   delay since last SR (DLSR)                  |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                 SSRC_2 (SSRC of second source)                | report
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ block
:                               ...                             :   2
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                  profile-specific extensions                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6-3: RTCP Sender-Report Format. Copied from RFC1889, pp. 23.

Another measurement method simply has the end system mark an RTP data packet every so often with an IP router alert option. Each RTP packet contains a payload type indication, which indicates the media encoding (such as G.711-encoded voice). For many low bit rate codecs, the payload type is associated with a fixed rate (*e.g.*, 64 kb/s for G.711), so that the router can make reservations based on that information alone. This mode has the advantage of trivial header parsing and fixed refresh intervals. (It also incurs the danger of increased packet delay variation and packet reordering since some RTP packets would traverse a routers “slow path”, while most would not.) According to the RTP specifications [24, 118], separate ranges of the payload type values have been set aside for audio and video, so that a router can assign RTP flows at different granularity: by session, by payload type value or by media class. To reduce the number of queues, a router may simply assign all voice traffic to a single high-priority queue, for example and just track the multicast destination and accumulated bandwidth for each session. However, this method does not work for dynamic RTP payload types, in which case, the sender may change the RTP data payload type in the duration of the session.

Similar to the RTP PT format, the IP DSCP (DiffServ Code Point) format allows routers to use the IP DSCP information [119, 120, 121] in RTP data packets to map them to the appropriate scheduler queue. This allows the router to keep track of the bandwidth allocated for each DSCP value, preventing over-commitment, yet avoids having to look up per-flow state for each packet. To prevent abuse by end applications, routers rather than end systems would be expected to set the DSCP value at each hop.

We had considered of supporting IntServ flowspecs in YESSIR. However, after more careful evaluation, we realized that the IntServ models are strictly based on the assumption that per-flow reservations must be established at *every* hop inside the network. In a partial reservation approach, such as YESSIR, it allows reservations to be made on some of the network links initially. It also allows end users to transmit data traffic at any time, *i.e.*, before, during and after reservation process, over the potentially partially reserved network links. Hence, we conclude that YESSIR would not operate well in an IntServ-enabled networking environment.

6.4.3 Error Handling and Killer Reservation

In YESSIR, a router does *not* generate error messages to the senders, nor does it try to automatically correct problems with error messages such as using RESVERR messages to create *blockade states* in an attempt to recover from *killer reservations* in RSVP. Instead, it has the option to insert error information into the RTCP SR message. It is up to the receivers to inform the senders about reservation failures via RTCP receiver reports. *RTCP*

SR's containing YESSIR reservation requests are always forwarded, even if unsuccessful.

We chose this approach for several reasons:

1. For high-bandwidth links, such as gigabit Ethernet, there is no reason to reserve resources for small data streams. In this case, a router should ignore YESSIR messages, and forward the requests downstream.

2. Managing resource over shared-media network such as Ethernet and Token-Ring networks is difficult. In this case, a router can insert a "reservation-undoable" flag in the error fragment of the RTCP SR message and forward it downstream.

3. This behavior is simple to implement. As shown in several RSVP implementations [67], the support for error message handling and associated blockade states is costly in terms of protocol processing, timer management and extra state storage.

4. Most importantly, as elaborated in Chapter 5, if network resources are not available during the first reservation attempt, there is always a possibility that reservations can be made when other reservations have terminated or during soft-state refresh times. There are many methods that routers can apply to recover from initial reservation failures. Hence, we believe that it is not appropriate to remove the reservation states and generate error messages when there is an admission control failure, nevertheless, the end users may need to be notified.

To illustrate our last point, we present an example here. In a heterogeneous network, a reservation request may fail for any number of reasons at a router. Unfortunately, such failures may also affect requests from other senders. Figure 6-4 demonstrates one of the *killer reservation* (or *reservation fragmentation*) effects. Two requests $Q1$ and $Q2$ (where $Q1 < Q2$) arrive at router Rt1. If $Q2$ arrives first and is accepted at Rt1, but rejected at Rt2, it could cause a smaller reservation $Q1$ to be rejected at Rt1 since the resource has been taken by $Q2$. As a result, neither request will enjoy any end-to-end QoS guarantee.

RSVP and ATM each solve this problem differently. If an ATM reservation cannot be accepted by a switch, that switch sends back a resource release message towards the sender to tear down the reserved resource at upstream nodes. RSVP generates RESVERR

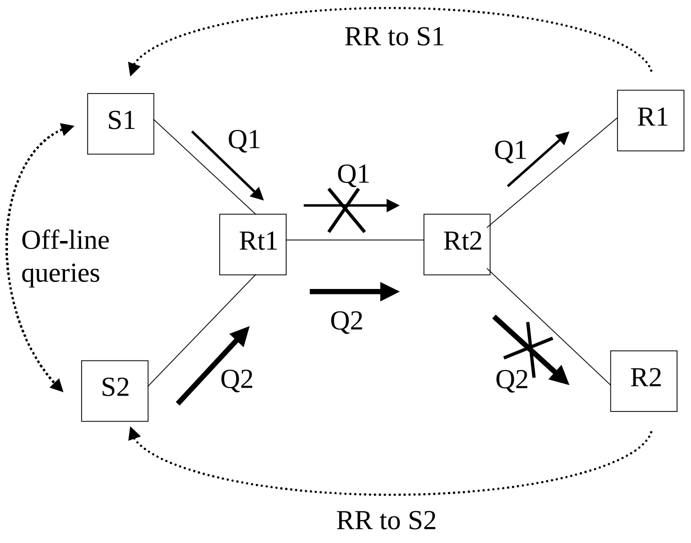


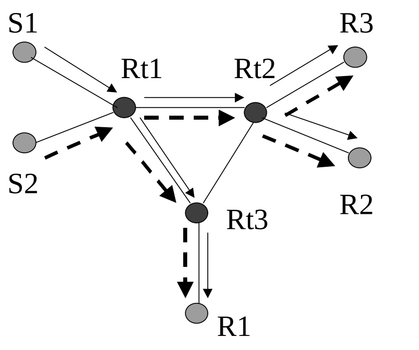
Figure 6-4: Problems due to resource contention

messages and creates a *blockade state* to allow smaller reservation requests to go through while blocking large requests. Unfortunately, blockade states are difficult to manage and incur high implementation complexity.

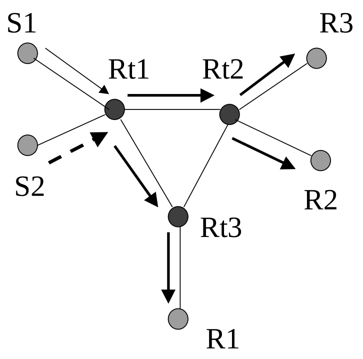
In YESSIR, partial reservations for both $Q1$ and $Q2$ will be made. However, senders receive an indication that the reservation was only partially successful and can then change or drop the reservation, clearing the way for other reservations to succeed.

6.4.4 Reservation Styles

YESSIR defines two reservation styles, individual and shared. In individual reservations, every sender in a RTP session has a resource reservation of its own. As shown in Figure 6-5 (a), router Rt1 receives reservation requests from both senders S1 and S2. After making a reservation, there are two separate reservations on links between Rt1 to Rt2 and Rt3. Depending on the amount of resources requested, RTP data streams from S1 and S2 may have different levels of reservation.



(a) *Distinct Reservation style*:
Reservations for S1are shown as in solid line; S2, in dotted line.



(b) *Shared Reservation style*:
At Rt1, after flow merging between reservation for S1 (solid line) and S2 (dotted line), a single reservation (thicker line) is made to Rt2 and Rt3.

Figure 6-5: Different reservation styles (S1 and S2 are senders, R1, R2 and R3 are receivers in a single multicast RTP session; Rt1, Rt2 and Rt3 are routers)

In a shared reservation, all senders within an RTP session share a single resource reservation in the network. As illustrated in Figure 6-5 (b), the links Rt1-Rt2 and Rt1-Rt3 have a single shared reservation. The amount of resources reserved on the link is the least upper bound (LUB) of the individual flow requests from S1 and S2. For example, if S1 and S2 request 10 kb/s and 15 kb/s of bandwidth, respectively, the shared bandwidth for link Rt1-Rt2 will be 15 kb/s. If there is a reservation failure, the reservation rejection information and the merged flow specification will be piggybacked in the RTCP sender report. Receivers will feed back the failure information and rejected reservation request to all participating members, including the senders. The senders can use these reports from receivers to adjust their requests. Flow merging issues will be addressed further in Section 6.5.1.

6.4.5 Other Considerations

Network Resource Advertising

In order to satisfy end-to-end service requirements, we adapted the OPWA (*One-Pass With Advertising*) scheme proposed by Shenker and Breslau [73] and described by Wroclawski [31] for YESSIR. Here is how it works in the context of YESSIR: each reservation request message carries a network monitoring fragment that consists of fields for hop counts, propagation delay, aggregated bandwidth and delay bounds. As SR messages traverse routers, this fragment will be updated at every hop. The receivers, upon reception of the SRs, will send the collected path resource information back to the senders in RTCP receiver reports. The senders can refer the path resource information to adjust their reservation levels by sending new requests.

Dynamic Reservation

An RTP session may not require a reservation for its whole duration. If reservations cost money, an application may well decide to only reserve network resources if best effort service proves unsatisfactory[2]. RTP-based applications supporting YESSIR can easily operate in this "reserve-when-needed" mode, as YESSIR reservation requests are coupled with RTCP messages. RTCP receiver reports have been designed to monitor traffic statistics. Senders can monitor receiver reports and only include a reservation request if a sufficiently large fraction of receivers indicate reception problems.

Network Security

RTCP and RTP data are tightly coupled. Thus, at a firewall, when a rule for a particular RTP data stream is defined, it will be automatically applied to the corresponding RTCP messages. Similarly, if a rule has been define to accept certain RTCP messages, the associated RTP data will be accepted as well. Supporting reservations across firewall filters is therefore greatly simplified.

[2]It obviously runs the risk that reservations will fail when the network is sufficiently busy to drop best-effort traffic.

Currently, YESSIR relies on security mechanisms at the IP layer [122] to provide authentication. If necessary, it would be easy to add an authentication facility to either RTCP or the YESSIR elements.

6.5 Description of YESSIR Algorithm

YESSIR on routers operate on two types of object. A "flow" object f has the following attributes:

f.source	traffic sender
f.destination	traffic receiver
f.ingress	ingress interface identifier
f.cleanup-timer	soft-state time-out timer
f.request-qos	requested QoS
f.nhops	a list of egress interfaces that require resource reservation

An "egress interface" object I has the following attributes:

I.egress	egress interface identifier
I.running-qos	operational QoS

Each YESSIR reservation flow can be uniquely identified by IP source and destination addresses and transport-layer port numbers[3]. Routers store source IP address and port number in f.source, and destination IP address and port number in f.destination. For multicast, each "flow" may require to reserve resources on multiple egress interfaces, thereby, f.nhops is a set of "egress interface" objects. To prevent looping, we keep track of both ingress and egress interface identifiers [4].

Routers can determine a flow's reservation type by simply checking if RTCP SR messages have an YESSIR extension attachment, as presented in Figure 6-2. If they do, routers

[3]All RTP/RTCP packets are running on top of UDP, that has a protocol number of 17 in the IP header.

[4]In BSD, this is the *if-index*. The router OS is responsible for making sure that the identifiers are unique and fixed at all time, even in hot-pluggable condition.

must operate in the *explicit* reservation mode. Otherwise, routers operate in *measurement-based* mode.

To operate in explicit reservation mode, routers need to abstract the following information from RTCP message M:

M.source	sender information (source IP address and port number)
M.destination	receiver information (destination IP address and port number)
M.command	Reserve or Tear-Down
M.style	Distinct or Shared reservation
M.refresh-cycle	soft-state refresh period
M.error	error information
M.opwa	One-Pass With Advertisement segment
M.qos	requesting QoS

The reservation setup details can be seen in the pseudo-code in Figure 6-6.

For measurement-based reservation, routers use a pre-configurated refresh timer, R, to manage soft-states. Figure 6-7 is the router algorithm for a measurement-based reservation, that interprets reservation level based on the timing and byte counters from RTCP SR messages. It needs to read the following information from RTCP message M.

M.5-tuple	from IP and UDP headers
M.timestamp	from RTCP SR
M.total-bytes	from RTCP SR

In addition, each flow object f requires to manage two new attributes:

f.last-timestamp	last received NTP timestamp
f.last-bytecount	last received byte counts

To delete distinct reservation flows, routers use the same procedures, as shown in Figure 6-8. In the section below, we will see that deleting a shared reservation flow can be somewhat more tricky.

```
1: on receiving RTCP SR message $M$, with YESSIR extension from interface $in$:
    if $\exists$ flow $f$ such that $f$.source $=$ $M$.source and $f$.destination $=$ $M$.destination
        if $M$.command = Tear-Down
            remove reservation from $f$;
            delete $f$ and exit;
    else
        create a new flow $f$;
        $f$.source $\leftarrow$ $M$.source;
        $f$.destination $\leftarrow$ $M$.destination;
    $f$.ingress $\leftarrow$ $in$;
    $f$.cleanup-timer $\leftarrow$ 3 $M$.refresh-cycle;
    $f$.request-qos $\leftarrow$ $M$.qos;
    query routing and find egress interfaces for $M$;
    for each egress interface $out$, such that $out \neq in$
        if $\exists$ egress $I \in f$.nhops, such that $I$.egress $= out$
            if $I$.running-qos $\neq$ $f$.request-qos
                update reservation on $I$;
        else
            create a new egress entry $I$ for $f$;
            $I$.egress $\leftarrow$ $out$;
            update reservation on $I$;
    forward $M$ downstream.

2: <update reservation on egress $I$ >:
    make/change reservation $f$.request-qos on interface $I$.egress;
    if succeed
        $I$.running-qos $\leftarrow$ $f$.request-qos;
    else
        apply retry algorithms to $f$;
        add failure code to $M$.error;
    update packet classifier from $f$.ingress to $I$.egress;
    update $M$.opwa.
```

Figure 6-6: YESSIR router algorithm - setting up a distinct explicit reservation

For clarity, we have omitted the details in message redundancy checking and reservation processing. In particular, in the measurement-based reservation, we need to consider the IP and UDP header overhead in the bandwidth approximation. The algorithms for

```
1: on receiving RTCP SR message $M$, without YESSIR extension from interface $in$:
    if $\exists$ flow $f$ such that $f$.source $=$ $M$.source and $f$.destination $=$ $M$.destination
        compute bandwidth usage from $M$ for $f$;
    else
        create a new flow $f$;
        $f$.source $\leftarrow$ $M$.source;
        $f$.destination $\leftarrow$ $M$.destination;
        $f$.request-qos $\leftarrow$ NULL;
    $f$.ingress $\leftarrow$ $in$;
    $f$.cleanup-timer $\leftarrow$ $3\ R$;
    $f$.last-timestamp $\leftarrow$ $M$.timestamp;
    $f$.last-bytecount $\leftarrow$ $M$.total-bytes;
    if $f$.request-qos $=$ NULL
        forward $M$ downstream and exit;
    query routing and find egress interfaces for $M$;
    for each egress interface $out$, such that $out \neq in$
        if $\exists$ egress $I \in f$.nhops, such that $I$.egress $= out$
            if $I$.running-qos $\neq$ $f$.request-qos
                update reservation on $I$;
        else
            create a new egress entry $I$ for $f$;
            $I$.egress $\leftarrow$ $out$;
            update reservation on $I$;
    forward $M$ downstream.

2: <compute bandwidth usage from message $M$ for flow $f$ >:
    new parameters: $\Delta_{time}, \Delta_{bytes}$
    $\Delta_{time} \leftarrow (M$.timestamp $-$ $f$.last-timestamp$)$;
    $\Delta_{bytes} \leftarrow (M$.total-bytes $-$ $f$.last-bytecount$)$;
    $f$.request-qos $\leftarrow (\Delta_{bytes}/\Delta_{time})$.

3: <update reservation on egress $I$ >:
    make/change reservation $f$.request-qos on interface $I$.egress;
    if succeed
        $I$.running-qos $\leftarrow$ $f$.request-qos;
    else
        apply retry algorithms to $f$;
    update packet classifier from $f$.ingress to $I$.egress;
```

Figure 6-7: YESSIR router algorithm - for setting up a distinct measurement-based reservation

```
1: on receiving RTCP BYE message $M$:
    if $\exists$ flow $f$ such that $f$.source $= M$.source and $f$.destination $= M$.destination
        remove reservation from $f$;
        delete $f$ and exit;
    update $M$.opwa.
    forward $M$ downstream.

2: on expiration of $f$.cleanup-timer:
    remove reservation from $f$;
    delete $f$.

3: $<$remove reservation from flow $f$ $>$:
    for each egress $I \in f$.nhops
        release resource $I$.running-qos;
        remove packet classifier;
        delete $I$ from $f$;
    apply retry algorithms if necessary.
```

Figure 6-8: YESSIR router algorithm - for terminating a distinct reservation flow.

measurement-based reservations using RTP PT and IP DSCP are similar (actually, simpler) to the one outlined in Figure 6-7.

When there is already a reservation Q in place for a particular user flow, if the user requests a larger reservation $Q' > Q$, and the resulting modified reservation fails by admission control at routers, the previous reservation Q must remain and cannot be denied. Here, we assume that there is no admission control failure in packet classifier, *i.e.*, the routers also have enough hardware/software classifier entries to accommodate per-flow reservations.

As we can see from the algorithms, there are several key differences in explicit and measurement-based reservation processing. In explicit reservations, routers can insert admission control failure, and network link information to the RTCP SR's, so that the end-users can detect the network link capacities, locate resource *hot spots*, and adjust reservation level dynamically. On the other hand, in measurement-based reservations, since routers derive resource requirements from user data packets directly, end-users hence have no control over reservation level. But end-users are free from entering reservation details.

Routers can switch between explicit and measurement-based mode at run-time. However, since it takes two consecutive RTCP SR messages to compute the bandwidth to be reserved, when routers are changing from the explicit mode to the measurement-based mode, the on-going reservations cannot be interrupted.

The routing interface depends in the flexibility and modularity of routing protocol implementation. We need to be careful that the reservation process simply sits on top of the routing process to setup reservation path, and does not alter the routing decision process. However, whenever there is a route change, routing process has the option to immediately inform the reservation process, so that proper adjustments can be applied in a timely fashion. When there is no routing upcall mechanism available, the reservation process can always rely on soft-state to gradually discover new routes.

Routers can apply various retry algorithms, including roll-back, fast retry and preempt, to get resources after an initial admission control failure. Details in those algorithms are in Chapter 5.

6.5.1 Flow Merging and Shared Reservation

In YESSIR, flow merging only takes place for shared reservations for routers that operate in the explicit-reservation mode. As discussed earlier, the merged flowspec is the least upper bound (LUB) value of the flowspecs from all participating senders. Here, we propose a best-effort approach to flow merging: when there is already a reservation in place, this reservation remains if a larger reservation request from another sender cannot be granted. As a result, all senders will have some fraction of their bandwidth reserved, though they may have different reservation requirements.

Figure 6-9 shows an example. S1 and S2 are the initial senders of a shared-reservation RTP session. The merged flowspec Q' is reserved inside the network, where $Q' = \text{LUB}(Q1, Q2)$. Later, a new sender S3 joins the RTP session and requests $Q3$ worth of resources. Router Rt2 tries to reserve the merged flowspec $Q'' = \text{LUB}(Q', Q3)$. Assume the reservation is successful and the new request Q'' is relayed to router Rt3. If Rt3 cannot reserve Q'',

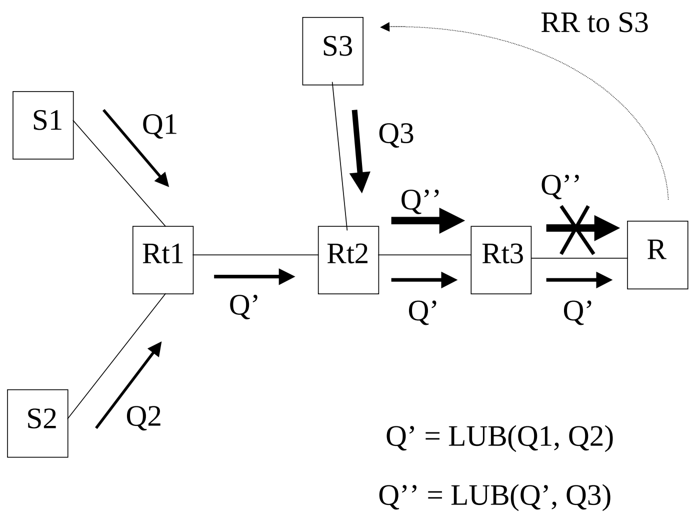


Figure 6-9: Flow merging for shared reservation.

it should continue to use the previous reservation Q'. Sender S3 will be informed about the last workable reservation Q' from receiver R via RTCP and will ultimately decide if it wishes to continue to participate in the session or whether it can lower its sending rate.

To support shared reservations, routers need to introduce a new "group" object g to maintain all the individual flows that share the same reservation. The object contains a list of "flow" objects (f's), as described in the previous section, in g.member. By definition, shared reservation is to have multiple flows that originate from different senders but go to the same destination using the same resource reservation. Therefore, the "group" object needs also to keep track of session, shared reservation level, and egress interface information.

g.member	a list of flows within the same session that share the same resource
g.destination	traffic receiver (unicast or multicast)
g.shared-qos	shared reservation amount
g.nhops	a list of egress interface that require reservation

The YESSIR extension for RTCP SR contains a shared reservation indicator so that routers can handle the flows using a different set of algorithms. Figure 6-10 and 6-11 are the router's algorithms to process shared reservations.

Compared to distinct reservation, processing shared reservations is a simple addition. The most challenging part is at reservation removal, where it requires to manage group members and shared resource with care.

6.5.2 Updating the Packet Classifier

As shown in Figure 6-12, when a YESSIR message is received, the YESSIR process queries the traffic control database for resource availability. If the resource is sufficient at the egress interface(s), the process updates the database and the scheduler.

According to the RTP profile [24], RTP data uses an even port number and the corresponding RTCP stream uses the next higher (odd) port number. Thus, during the parsing of RTCP messages, RTP data packet information including the IP source and destination addresses, port numbers and protocol type can be learned automatically.

After the router successfully sets up the scheduler, it inserts RTCP's IP source and destination address, protocol type (presumably, UDP), and the corresponding RTP data port numbers into the flow table. When RTP data packets are received, the packet classifier filters on the IP and UDP headers and forwards the packets to the scheduler.

6.6 YESSIR Implementation and Results

As a lightweight signaling protocol, YESSIR is designed to provide resource reservation to real-time flows that use RTP. It operates in two operating modes, *explicit*, and *measurement-based*, and supports both *distinct* and shared reservation styles.

To reserve network resource for RTP data streams, YESSIR needs to insert IP Router-Alert option in RTCP Sender Report messages. Unfortunately, not all the routers support the IP Router-Alert option.

In Internet protocol development, message parsing is *not* really the focus, rather, state management and task management are the key considerations. YESSIR is no exception.

```
1: on receiving RTCP SR $M$, for *shared* reservation from interface $in$:
    find/create group $g$, such that $g$.destination $=$ $M$.destination;
    **if** $\exists$ flow $f$ such that $f$.source $=$ $M$.source and $f$.destination $=$ $M$.destination
        **if** $M$.command = Tear-Down
            remove reservation from $f$;
            delete $f$ and exit;
    **else**
        create a new flow $f$;
        $f$.source $\leftarrow$ $M$.source;
        $f$.destination $\leftarrow$ $M$.destination;
    $f$.ingress $\leftarrow$ $in$;
    $f$.cleanup-timer $\leftarrow$ 3 $M$.refresh-cycle;
    $f$.request-qos $\leftarrow$ $M$.qos;
    $g$.shared-qos $\leftarrow$ ($g$.shared-qos) LUB ($f$.request-qos);
    add $f$ to $g$.member;
    query routing and find egress interfaces for $M$;
    **for** each egress interface $out$, such that $out \neq in$
        **if** $\exists$ egress $I \in g$.nhops, such that $I$.egress $= out$
            **if** $I$.running-qos $\neq$ $g$.shared-qos
                update reservation on $I$;
        **else**
            create a new egress entry $I$ for $g$;
            $I$.egress $\leftarrow$ $out$;
            update reservation on $I$;
    forward $M$ downstream.

2: $<$update reservation on egress $I$ $>$:
    make/change reservation $g$.shared-qos on $I$.egress;
    **if** succeed
        $I$.running-qos $\leftarrow$ $g$.shared-qos;
    **else**
        apply retry algorithms to $f$;
        add failure code to $M$.error;
    update packet classifier from $f$.ingress to $I$.egress;
    update $M$.opwa.
```

Figure 6-10: YESSIR router algorithm - for setting up a shared reservation

```
1: on receiving RTCP BYE message $M$:
    find group $g$, such that $g$.destination $=$ $M$.destination;
    if $\exists$ flow $f$ such that $f$.source $=$ $M$.source and $f$.destination $=$ $M$.destination
        remove reservation from $f$;
        delete $f$.
    update $M$.opwa;
    forward $M$ downstream.

2: on expiration of $f$.cleanup-timer:
    find group $g$, such that $g$.destination $=$ $f$.destination;
    remove reservation from $f$;
    delete $f$.

3: <remove reservation from flow $f$ >:
    remove $f$ from $g$.member;
    remove packet classifier;
    if $g$.member $= NULL$
        for each egress $I \in g$.nhops
            release resource $I$.running-qos;
            delete $I$ from $g$;
        remove $g$;
    else
        for each flow $f_i \in g$.member
            $g$.shared-qos $\leftarrow$ ($g$.shared-qos) LUB ($f_i$.request-qos);
        for each egress $I \in g$.nhops
            update reservation on $I$;
    apply retry algorithms if needed;
```

Figure 6-11: YESSIR router algorithm - for terminating a shared reservation

As we can see from the previous section, the YESSIR processing algorithm is very straightforward, and can setup reservations in a single path. However managing flow, interfaces and group entries *efficiently* can be challenging.

In this section, we first describe the implementation of the IP Router-Alert option in a FreeBSD system, and the method we used to manage reservation flows. We have implemented a version of YESSIR on FreeBSD, and implemented both YESSIR and RSVP on IBM 2210 router platform. We will show the performance results in the rest of the section.

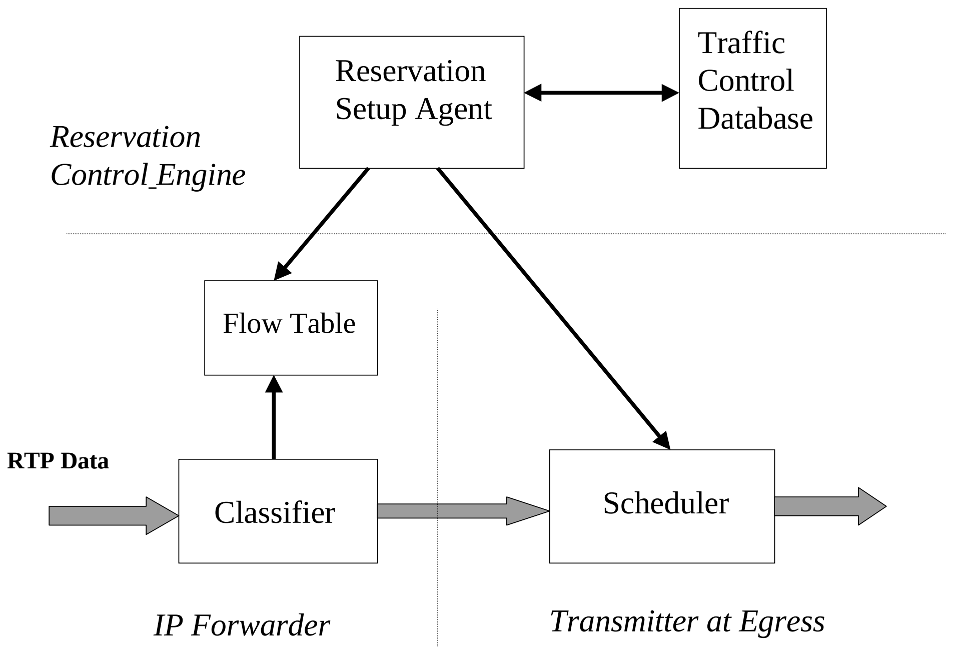


Figure 6-12: A router model for reservation support.

Figure 6-13 is an outline of YESSIR reservation sequence. We will use its terminology in our performance benchmarking.

6.6.1 BSD Kernel Extension

Current UNIX system have no clean mechanism to intercept IP option packets, including router alerts, through the socket interface. Thus, we designed and developed a new socket family, `PF_IPOPTION`, on FreeBSD . To receive IP packets with option type, *type*, a user needs to open a raw socket:

```
socket (PF_IPOPTION, SO_RAW, type);
```

For example, the routers can capture RTCP (YESSIR) messages with IP Router Alert option as the following:

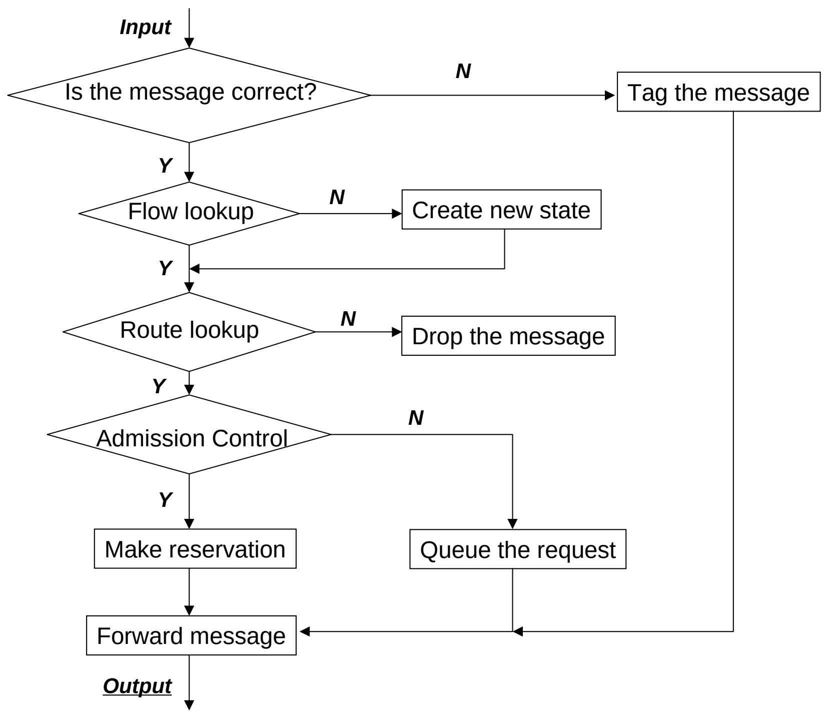


Figure 6-13: YESSIR reservation processing flowchart at routers.

```
int sock, on = 1;


sock = socket (PF_IPOPTION, SO_RAW, IPOPT_RA);
setsocketopt (sock, IPOPT_OP, IP_RECVRTCP, &on, sizeof(on));
```

The user process can receive a YESSIR message as the following:

```
struct iovec iov;               /* provide data buffer info */
struct cmsghdr *cmsg;           /* pointer to control message */
char *packet = (char *)malloc(MAX_PKT_SIZE);
char *ctrl = (char *)malloc(MAXCTRLSIZE);
...
iov.iov_base = (char *)packet;
iov.iov_len = MAX_PKT_SIZE;
```

```
msg.msg_iov = &iov;
msg.msg_control = ctrl;
msg.msg_controllen = MAXCTRLSIZE;
...
len = recvmsg(sock, &msg, 0);
for (cmsg = CMSG_FIRSTHDR(&msg); cmsg != NULL;
  cmsg = CMSG_NXTHDR(&msg, cmsg)) {
      if (cmsg->cmsg_type == IP_RECVIF)
           if_index = CMSG_IFINDEX(cmsg);
      ...
}

rtcp_read(packet, len, if_index);  /* process RTCP message */
    ...
```

To improve packet processing performance, we programmed the kernel to process the IP option packets as soon as possible, as shown in Figure 6-14. To improve the socket I/O performance, we allocate one PCB (Protocol Control Block) chain for each IP option type, as oppose to the traditional approach, that is, to use a single PCB chain to keep track of all socket calls [123].

6.6.2 Reservation State Management

We used hash tables to manage the reservation states in our implementation. The motivation behind using hashing comes from the observation that any RTP session in the network can be *uniquely* identified by its IP source address, *SA*, and destination address, *DA*, and its UDP source port, *SP*, and destination port, *DP*. In our implementation, the hash key for an RTP flow f_i is a simple summation, $k_i = SA_i + DA_i + SP_i + DP_i$. Our goal was to support up to 1,000 flows efficiently, so we selected the hash table size, M, to be a prime number, 1,537, to reduce the chance of hash collisions [124]. To solve the potential hash collision problem, we put all the flow entries that hash to the same slot in a linked list.

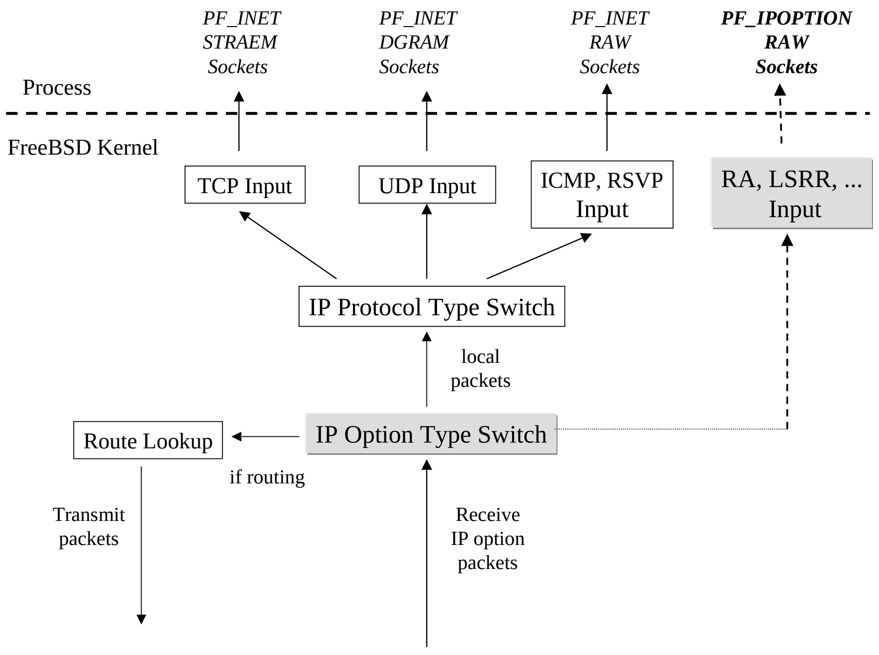


Figure 6-14: Relationship of IPOPTION processing to rest of FreeBSD kernel. The shaded box is where the IP option packets to be processed. The implementation supports all defined IP options. RA is the Router-Alert option, and LSRR is the Loose Source Routing Record option.

Our hash function is

$$h(k_i) = k_i \text{ mod } M.$$

Figure 6-15 illustrates the reservation state handling in our implementation. There are three tables: hash index table, free entry bucket and flow entries. When a reservation request for flow f_i is received, we perform the hash function to find a hash slot, k, in the hash index table. After getting a flow entry from the free entry bucket, we copy the reservation data into the entry, and insert it into the linked list that is hanging off the hash slot, k.

A collision occurs if a new flow f_j arrives and hashes to the same slot as f_i. We simply insert the new entry into k's list behind the flow entry for f_i. Obviously, too many collisions

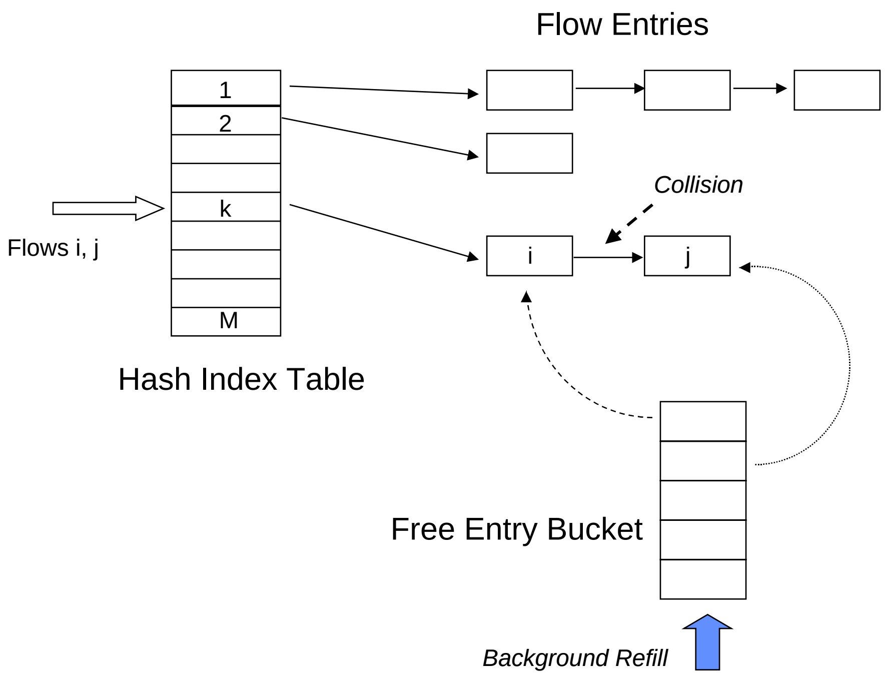


Figure 6-15: An example of reservation state management in our implementation.

will cause poor performance. (More sophisticated dynamic hashing schemes can limit the depth of the linked list.)

To improve memory usage and simplify debugging, we have designed a free entry bucket (FEB) table. At system initiation time, we pre-allocate a small number of entries into the FEB. During flow processing, if the number of free entries is less than a threshold number, we will allocate another chunk of entries into the FEB. If there are too many free entries, the process will free the extra entries.

The hash table and memory management take about 1,400 lines of C code to implement.

6.6.3 FreeBSD Implementation

We have implemented YESSIR on FreeBSD version 3.5. The latest version of the YESSIR implementation requires about 6,000 lines of C.

We tested and measured the implementation on a 700 MHz Pentium PC with several Ethernet interfaces. We have modified a RTP software testing package, rtptools [125], to generate RTCP messages with IP Router Alert option from end users.

Number of flows in the router	flow creation time (μs)
50	6.4 $\pm$ 1.35
100	6.3 $\pm$ 1.25
200	6.3 $\pm$ 0.95
500	6.2 $\pm$ 1.32
800	6.4 $\pm$ 0.97
1,000	6.1 $\pm$ 0.99
2,000	7.3 $\pm$ 1.25
5,000	7.1 $\pm$ 1.20
8,000	8.1 $\pm$ 0.99
10,000	8.0 $\pm$ 1.56

Table 6.1: Hash table performance with collision resolution by chaining. The hash table size is 1537.

We first examined the efficiency of the QoS state management with hashing. Provided that the hash table size is 1,537, we expect that the flow entry searching and creation time would be more or less the same when there are less than 1,000 flows in the system. As the number of the flows increases, more hash collisions will occur.

To verify this, we generated 10,675 new reservation flows from multiple sources. On the router, we had recorded the flow entry creation time on each flow. To ensure measurement accuracy, we shut off the refresh timers during the test. The results are shown in Table 6.1. As expected, the processing time is constant if the number of flows is below 1,000 and gradually rises above that threshold.

We measured the time for processing a YESSIR reservation message. The measurement was taken both at user space and in the kernel. During the measurement, we generated YESSIR messages used for measurement-based reservation. As shown in Table 6.2, the overall processing time in the user space is about 46 μs. However, the time between when the packet is received from the device driver until it is sent to the device driver in the kernel is 98 μs, which is approximately half the processing time the packet spent in the kernel.

Description	Processing time (in μs)
Message integrity checks	6.6 $\pm$ 0.52
Flow lookup/creation	6.5 $\pm$ 0.53
Route lookup	27.1 $\pm$ 1.29
Admission control (call for reservation)	6.1 $\pm$ 0.57
Kernel I/O	97.8 $\pm$ 8.24

Table 6.2: Timing for single measurement-based YESSIR flow on a 700 MHz Pentium.

6.6.4 YESSIR and RSVP Performance Comparison

We have implemented both RSVP and YESSIR on the IBM Common Router Architecture software platform. Both implementations have similar data structures and coding style, and share the same set of data processing routines. We measured the various costs associated with RSVP and YESSIR on a router. The measured router was the IBM 2210 Nways Multiprotocol Router, which is based on a Motorola 68040 processor with a bus speed of 32 MHz. Processing times were measured by reading clock ticks from the timer register of the processor that has a timing resolution of 31.25 ns per tick. We divided the times into categories so that we can have somewhat loose objective comparison between RSVP and YESSIR.

The underlying routing protocol used in our experiment is OSPF, and it updates the routing table in case of route changes. A route query operation is a straight forward route-table look-up. All RSVP flows are set up as controlled-load, fixed-filter style, and encapsulated in IP with the router-alert option. YESSIR messages use the RTP PT (Payload-Type) format, the individual reservation style, and are encapsulated in RTCP, UDP and IP with the router-alert option. Since the packet classifier and scheduler are implemented differently depending on the physical network interface, but are the same for RSVP and YESSIR, we chose to bypass them in our tests. Data collected here only reflects the RSVP and YESSIR control path behavior.

Tables 6.3 and 6.4 present the protocol processing overheads of setting a new RSVP or YESSIR flow. Table 6.5 and 6.6 show the processing overhead of refreshing a flow.

Code Section	Time (μs)	% of Total
PATH Processing:		
RSVP flow lookup/creation	410.51 $\pm$ 7.51	37.1%
Route lookup	40.61 $\pm$ 1.84	3.7%
Send PATH downstream	283.16 $\pm$ 3.85	25.6%
RESV Processing:		
RSVP flow look-up	11.03 $\pm$ 0.43	1.0%
Admission Control	126.35 $\pm$ 1.10	11.4%
Flow merging and forward RESV	234.14 $\pm$ 3.27	21.2%
Single RSVP flow setup overhead	1,105.80 $\pm$ 9.47	100%

Table 6.3: Router processing overhead for a new RSVP flow.

Code Section	Time (μs)	% of Total
Flow lookup/creation	41.40 $\pm$ 1.24	11.6%
Route lookup	38.43 $\pm$ 2.00	10.8%
Admission Control	23.33 $\pm$ 0.38	6.5%
Send message downstream	253.53 $\pm$ 0.74	71.1%
Single YESSIR flow setup overhead	356.68 $\pm$ 2.84	100%

Table 6.4: Router processing overhead for a new YESSIR flow.

From the measurement shown in Tables 6.3 and 6.4, we observe that a router can set up a new reservation flow with YESSIR three timers faster than if it uses RSVP. On soft-state refresh, the YESSIR processing overhead is nearly 50% less than that of RSVP.

Comparing all these times, we see that the overhead of constructing and sending a message is about 250 μsec. This includes getting a new buffer [5], copying data and scheduling

[5]Alternatively, we can simply modify the received packet instead of getting a new buffer. However, RSVP and YESSIR are designed to support multicast flows, where multiple buffers may be required to forward a

Code Section	Time (μs)
On receive:	
PATH lookup	30.39 ± 0.76
Route lookup	37.94 ± 1.99
RESV lookup	11.01 ± 0.35
Admission Control	44.05 ± 1.39
Timer routine:	
Send PATH Refresh	262.02 ± 10.20
Send RESV Refresh	239.06 ± 3.44
Single RSVP flow refresh overhead	624.46 ± 12.26

Table 6.5: Router processing overhead for RSVP refresh message.

Code Section	Time (μs)
On Receive:	
YESSIR flow lookup	19.31 ± 0.69
Route lookup	39.93 ± 0.38
Admission COntrol	24.49 ± 0.31
Forward YESSIR downstream	252.56 ± 2.00
Timer routine:	
YESSIR flow checking	8.03 ± 0.73
Single YESSIR flow refresh overhead	344.32 ± 1.88

Table 6.6: Processing overhead for YESSIR refresh message.

for transmission. RSVP requires to send two messages to setup a flow, while YESSIR takes only one message.

The packet transmitting overhead becomes more critical during soft-state refresh. YESSIR

single packet at a router.

relies on end-to-end soft-state refresh, that is, end users periodically transmit RTCP SR's with IP-alert option to maintain the flows inside the network. As a result, a YESSIR refresh message takes about 344 μs to process. RSVP uses hop-by-hop soft-state refresh mechanism. A RSVP router in the network is required to get buffers, construct refresh messages and send them both upstream and downstream. As shown in Table 6.5, refreshing a RSVP flow takes about 624 μs. Even worse, transmitting RSVP refresh messages takes place at timer interrupt level, which locks up the memory bus during the processing, thus stalling the packet forwarding loop. If a router maintains a large number of RSVP flows, its packet forward performance can be seriously degraded due to long timer interrupts. YESSIR, on the other hand, uses the timer to check the flow lifetime only, and therefore takes far less time in each timer interrupt (approximately 8 μs).

The times reported above suggest accessing packet memory at router can be expensive. Creating a new flow entry and updating reservation information require extensive message parsing and copying. Given that YESSIR's PT flowspec is far more smaller (one word in the message) than RSVP's IntServ-format Sender-Tspec and Flowspec objects, we observed that the cost for creating a new YESSIR entry is nearly ten times less than that for RSVP, and the cost for updating reservation data in YESSIR is five times less than that for RSVP.

The above data was collected on an edge router platform, where heavy control message processing can directly impact packet forwarding. However, this is not the case for some of the large routers, where data packets and control messages are processed on separate hardware. On the other hand, as shown in studies such as [20], large amounts of control messages can indeed effect the performance of large routers in WAN. Hence, it is important in reduce the complexity and the frequency of network control messages in all networking environments.

6.7 Related Work

Almesberger *et al.* [126] proposed a lightweight resource reservation solution, SDP. For a traffic sender that wishes to make a reservation, it starts by sending data packets marked with a *request* flag to the destination. The destination collects the traffic statistics, and periodically feeds it back to the sender. The sender therefore can estimate the amount of the reservation that the network can accommodate. It sends compliant packets marked with a *reserved* flag. The network routers will always accept the packets that are marked with *reserved* flag. In this solution, the routers rely on end-users to be network friendly, and not to exceed their “reserved” rates. There have been several other proposals [127, 115, 116, 128] that are based on variations of measurement-based admission control.

Feher *et al.* [129] proposed a lightweight reservation mechanism called Boomerang, where end users send reservation requests and refresh messages (in ICMP) to set up and maintain reservations. The reservations are initiated from the sender’s direction. Instead of storing reservation states to keep track of each flow, Boomerang relies on refresh messages to keep reservations alive at routers. The objective is to reduce the router’s processing and storage overhead.

6.8 Conclusion

Resource reservation is useful for supporting continuous-media services over the Internet. The question is: at what cost?

RSVP was designed to support resource reservation in the Internet, however, it has two significant problems: *complexity* and *scalability*. The former results in large message processing overhead at end systems and routers. The latter implies that the sheer number of reservation flows at a router can be too large.

YESSIR is designed to address the *complexity* problem. It provides a way to simplify the reservation processing and therefore reduce associated overhead at routers. The YESSIR approach has the following important features:

- It is sender-initiated, and supports one-pass reservation mechanism to simplify processing at routers.
- To speed up flow establishment, YESSIR allows partial reservations and support reservation retries.
- Both distinct and shared reservation styles can be supported efficiently in YESSIR.

- YESSIR takes advantage of the close relationship between RTP and RTCP packets for easy packet classification and firewall support.
- It uses soft state mechanisms to reliably and responsively maintain reservation states.

From evaluating our implementation results[6], we believe that with proper protocol design and implementation, routers can support a large number of user flows, while providing admission control.

[6]The source and object code for the YESSIR FreeBSD implementation and PF_IPOPTION socket family is available at http://www.cs.columbia.edu/˜pingpan/software/.

Chapter 7

BGRP: An Inter-Domain Reservation Protocol

7.1 Introduction

As discussed in Chapter 4, network providers have been using reservation signaling mechanisms, such as RSVP-TE [97], to setup intra-provider VPN's and dynamically provision network resource. Unfortunately, these signaling protocols have the potential reservation state "explosion" problem.

We propose a *hierarchical reservation model* in Chapter 4 to solve the reservation state scalability problem. End-user applications can trigger some lightweight signaling, such as YESSIR, to request network resources. At the network-level, providers need to be able to establish long-lasting, secure and robust reservations among each other. Application-layer reservations can readily aggregate into these provider-level reservations at network boundary.

In this chapter, we will propose a provider-level protocol, called the Border Gateway Reservation Protocol (BGRP), that has two significant scaling properties: first, BGRP overhead scales *linearly* with the size of the Internet. BGRP is designed to work with a small number of reservation entities: the overhead of the BGRP protocol is proportional to the number of Internet *domains* (*i.e.*Autonomous Systems (AS)).

The chapter is organized as follows. In Section 7.2.1, we outline the design objectives for inter-domain reservation signaling. Section 7.3 describes our design choices that can achieve the desirable aggregation gains in today's Internet environment. We present the protocol in Section 7.4, and propose several performance enhancements in Section 7.5. The scaling benefits of BGRP are evaluated in Section 7.6. We summarize our investigation in Section 7.8.

7.2 Requirements and Network Environment

7.2.1 Requirements and Assumptions

BGRP is based on the following assumptions and requirements:

Message processing: The cost of processing reservation messages depends on the complexity of handling each message and the frequency of reservation messages. Instead of setting up reservations across domains as application flows arrive, we rely on pre-computed reservations made in advance. This also implies that each reservation will last for relatively long period of time (typically hours or days).

Intra-domain vs. inter-domain reservation: It is desirable that each domain can manage its own network resources and enforce its own internal traffic engineering policies. This implies that a domain only reveals simple delivery commitments to its peering domains in terms of bilateral agreements. The inter-domain reservation then uses these delivery commitments to establish a reservation path through multiple domains. Each domain sets up transit reservation flows using its preferred intra-domain reservation mechanism.

Security: Regional ISP's subscribe to transit services from large backbone providers. Not only do the backbone providers have to guarantee adequate network resources to accommodate the user traffic demands from the regional ISP's, they also must protect customer's identity and traffic usage data. When two backbone providers establish the private peering, they must be sure that the peering information will not leak to the public. In the design of a inter-domain reservation protocol, it is critical to hide the end-user's identity, reservation demands and resource usage during the reservation setup.

State maintenance overhead: Routers need to maintain and store both reservation control state and packet forwarding state. To reduce the former, we need to be able to aggregate reservations. To reduce the latter, we rely on Diff-Serv [59] to eliminate per-flow queuing and processing, so that the number of queues is likely to be no larger than a few dozen. We need to be aware here that when setting up MPLS LSP's, reservation states may not be associated to the actual link resource (such as bandwidth), rather we need to manage upstream and downstream MPLS labels.

Bandwidth overhead: The bandwidth consumed by setting up reservations should be small compared to the link bandwidth, both in steady state and with routing transients. This bandwidth overhead is typically proportional to the number of flows kept in routers, and thus minimizing the state storage overhead helps here.

Routing interface: Reservation protocols rely on routing information to set up reservations following the data forwarding path, and must not interfere with route aggregation and effect routing protocol scaling properties. During route changes, routing protocols have the option to inform reservation protocols, which in turn will fix the effected flows. Otherwise, reservation protocols that employ soft-state to manage flows can periodically query the routing table to determine if the flow routing paths have changed.

7.2.2 Network Environment

Internet backbone consists of a number of domains, each of which has at least one border router (BR). From BGP, each BR learns about the other BR's within its own domain, and the directly connected BR's in the adjacent domains. Through out-of-band means, the BR's know of *bilateral* (or *multi-lateral*) agreements with the peering domains.

Typically, the bilateral agreement specifies the inter-carrier policy information such as route filtering and route preference [130]. In the future, we envision that the bilateral agreement may also include policies for QoS guarantees between peering domains. It is worth noting here that the bilateral agreement applies only between two adjacent domains, and network users may not have the knowledge or the certainty which downstream domains their traffic will traverse.

As being advocated in the IETF Traffic Engineering Working Group, the ISP's use RSVP-TE or CR-LDP to set up border-to-border (or edge-to-edge) intra-domain "virtual" trunks between border routers. At each NAP or POP, the ISP's set up similar "virtual" trunks to interconnect with external domains. The goal here is to optimize the use of network resource and traffic performance. Example of "virtual" trunks includes Frame Relay, ATM, MPLS and DiffServ.

7.3 Design Choices

7.3.1 Two-Pass Reservation Model

In Chapter 5, we have compared one-pass and two-pass reservation models. The one-way model is more efficient since it combines the *path pinning* with the *reservation process*. However, in inter-domain reservation signaling, an one-way model may not work well due to the following reason: when a requestor starts to set up an inter-domain reservation, it does not know the reservation's destination domain information at initiation time. In fact, to satisfy bilateral and security requirements, any provider along the reservation path can terminate the reservation.

For example, we need to setup a reservation between provider A and provider B, passing through provider C and D. Assume that provider D is not allowed to distribute A's reservation requests and routing information beyond its network. When A initiates a request toward B, D can intercept the message upon reception and complete the reservation between A to D. Based on other mechanisms, D can setup another reservation to B. Eventually, D can link two distinct reservations together. In this case, though there is a reservation between A and B, A may not be aware of the existence of the second reservation, nor the "real" destination of its reservation.

One may argue that QoS routing protocols would be sufficient for finding the path. Routers can first query QoS routing and then setup the reservations with an one-pass protocol. In an *inter*-domain environment, however, we believe that the reservation path depends as much on ISP policy as on resource availability. Since each ISP only advertises its resource allocation policy to its immediately adjacent peers in the form of bilateral agreements, a resource user may have to actively probe the network to determine the edge-to-edge routing path for its reservation.

Hence, we believe that in the inter-domain environment, a two-pass reservation model is a preferred choice, and reservation requestors need to go through a *path discovery* path before the actual reservation.

7.3.2 Sink Tree Approach

As discussed in Chapter 4, the root for the N-square problem is that routers have to maintain both source and destination information to prevent persistent looping caused by multicasting. In today's backbone, however, most of the traffic is unicast traffic. Plus, we can "tweak" multicast routing protocols to force packets to be sent on sender shared trees. Hence, we can relax the requirement of keeping both source and destination on routers.

By observing the MAE-West NAP traces in Chapter 4, we have derived the following: *if we set up reservations based on either source or destination AS's or network prefixes, we can readily keep the reservation count at a sustainable level (in the range of less than 10,000).*

Whether a simple hop-count metric is used or more sophisticated metrics are employed, most unicast routing algorithms determine shortest paths. In networks where the shortest paths are unique, the principle of optimality guarantees that the shortest paths *to* any *destination* form a tree; the shortest paths *from* any *source* are also guaranteed to form a tree. If there are multiple shortest-length paths, however, the existence of trees depends on the tie-breaking rules in the routing algorithm.

Today's inter-domain routing protocol, BGP [14], establishes "virtual edges" by using reachability as a definition for the existence of a link in the graph. In case of paths of equal weight, current practice dictates that the border routers forward all packets over only one path. This practice guarantees that routing always follow sink trees. Thus, *if reservations are made along routes chosen by the BGP routing algorithms, it is natural to aggregate these reservations along the sink trees formed by routing.*

This approach has a very desirable scaling property: the routers only need to remember the reservation *sinks*. We can further improve the scalability by allowing only the BGP border routers to participate in the reservation process.

7.3.3 How to Create Sink Trees?

There are many ways to establish reservation sink trees inside the backbone. One solution is that reservation senders query a centralized database to get the precise routing path prior to each reservation. A similar idea has been introduced in the RSVP TE extension [97], in which case an Explict-Route Object (ERO) can be computed and queried by the reservation senders before each request. However, this method may not be applicable in inter-domain environments. First, it is not clear how the centralized database can be managed among multiple providers. Further, it is doubtful that the centralized database can keep track of all the network topology and resource changes from multiple provider networks, and compute the “best” routes for the reservations.

Another solution is to allow the routing protocols, in this case, BGP, to set up sink-trees at route advertising and RIB (Route-Information-Base) processing time. However, this approach has scalability problems. Routing aggregation is designed to reduce the total number of route entries that routers have to manage. By default, all border routers are required to aggregate routes whenever possible. As a result, BGP routing updates normally do not travel many provider networks before they are aggregated. However, if reservation requests are coupled with routing information, routers will have to stop the route aggregation in order to deliver reservation messages between reservation senders and receivers.

Here we propose a two-pass distributed reservation solution. The reservation senders send query messages to the network. The queries will follow the data path and are delivered across BGP hops. At each BGP hop, a routing path is selected based on the bilateral agreement. However, the routers do not pin down the reservation path and do not store the query data.

The reservation receivers keep track of all the queries, and construct sink-tree graphs from the information. The receivers send reservation request messages upstream to set up the actual reservations. At each border router, the router aggregates the reservations for each sink tree.

7.4 The Border Gateway Reservation Protocol (BGRP)

The Border Gateway Reservation Protocol (BGRP) is an inter-Autonomous System reservation protocol. It sets up aggregated reservations over multiple Autonomous Systems (AS). The reservation originators and the terminators are at the BGP-speaking border routers. Reservation aggregation is at the AS level. Only BGP-speaking routers in the network participate in the reservation process.

All traffic destined for a particular AS can be aggregated in a sink-tree fashion. Since backbone routers only maintain the sink tree information, the total number of reservations at each router scales, in the worst case, linearly with the number of AS's in the Internet.

BGRP runs over TCP and thus eliminates the need to implement fragmentation, retransmission and sequencing. BGRP uses "soft-state" to manage reservations, to protect against events such as link failure.

BGRP is not involved in setting up and managing intra-domain reservations. We envision that the ISP's may use the RSVP-Traffic Engineering (TE) extension, or other means to manage internal network resources. BGRP only becomes useful when reservations need to be established across multiple AS's.

The current version of BGRP does not provide support for multicast traffic.

7.4.1 Terminology

As shown in Figure 7-1, each domain has an unique AS (autonomous system) number, and can exchange user traffic with its peers. Each domain is under a single common administration. A domain can be classified as either a *stub* domain or a *transit* domain. Any path

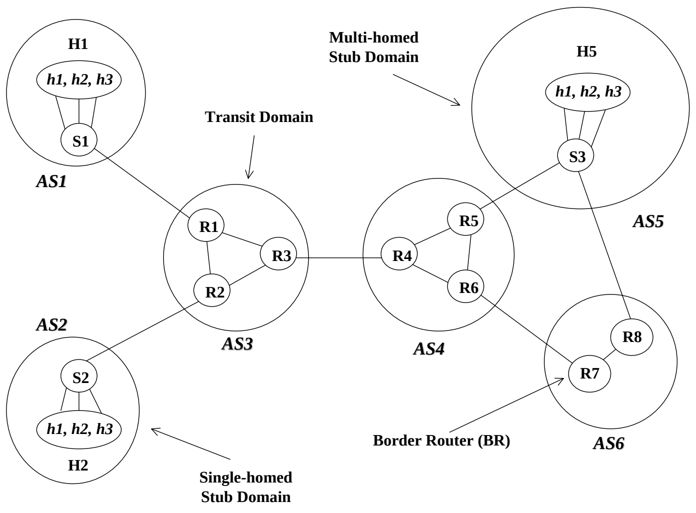


Figure 7-1: Example of Internet domains. There are two types of stub domains: *single-homed* stub domains connect to the backbone at a single point, *multi-homed* stub domains at several points.

through a stub domain will either originate or terminate at a router in that domain; transit domains do not have this restriction.

A domain connects to a number of other domains via *border routers* (BR). We assume all border routers use BGP4 for inter-domain routing. For simplicity, we only consider the EBGP (External-BGP) border routers[1] at present time. We define $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ as the set of border routers in transit domains, $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ as the set of border routers in stub domains, and $\mathcal{H}_i = \{h_1, h_2, \ldots, h_j\}$ as the set of end hosts in AS$_i$. $\mathcal{H} = \cup\mathcal{H}_i$ comprises all hosts in the network. For simplicity, we only illustrate the cases where inter-domain reservations originate and terminate at routers in $\mathcal{S}$. In reality, a transit router R_i could also play the role of a source or sink router for end users in its domain. Moreover, while in reality there are likely to be multiple routers in a domain between border routers, they do not participate in our inter-domain reservation protocol and are thus not shown in

[1]When BGP is used to exchange routes that belong to different domains, it is called EBGP (External-BGP), as oppose to IBGP (Internal-BGP) that manages routes within the same domain. IBGP uses route reflectors [19] and confederations [17] to solve the internal route scaling problems. We will consider the interface between BGRP and reflectors and confederations in future studies.

the figures.

We define the directional terms *upstream* and *downstream* with respect to the direction of data flow. The traffic direction from source to destination is *downstream*; destination to source, *upstream*. A reservation *sender* is an upstream border router that originates reservation messages. A reservation *receiver* is a downstream border router that terminates reservations.

BGRP operates only between border routers. We shall use the term *hop* to denote the path between two “adjacent” BGP border routers participating in the reservation process.

Reservation aggregation occurs if multiple reservations coming from different sources but going toward the same destination can be “added” together to create a single reservation.

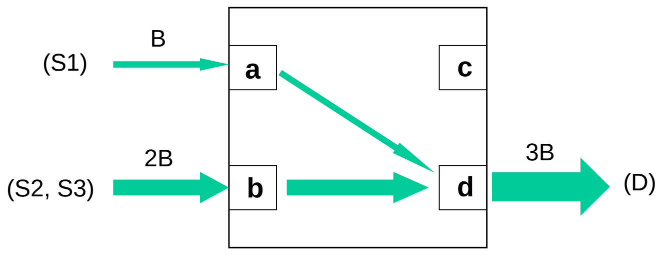


Figure 7-2: BGRP example at a router: bandwidth reservation aggregation from sources S1, S2 and S3 to destination D.

Figure 7-2 illustrates the concept of bandwidth reservation aggregation. (a), (b), (c) and (d) are the interfaces at a router. Reservations from data source S1, S2 and S3 are all going to destination D. In the example, each source needs B units of bandwidth. At (d), all individual reservations are aggregated together. The amount of reservation at (d) is $3B$ as a result.

7.4.2 Outline of Protocol Operation

BGRP defines several messages: Probe, Graft, Error, Refresh, and Tear. Probe and Graft messages specify reservation parameters such as bandwidth. Here we assume that bandwidth is the only reserved resource and that bandwidth reservations are additive. In practice, however, other resources such as buffers could be reserved.

Reservation sources initiate Probe messages to discover the reservation path and the network resource availability. The Probe messages travel downstream and traverse BR's hop-by-hop until reaching the destination domain. Border routers rely on BGP routing information (BGP Next-Hop attributes) and bilateral agreements to forward Probe messages. Probe messages must travel the path that is used for actual reservation.

Each Probe message consist of a desired resource amount and destination network information. Probe messages collect routing information along the reservation path, similar to *IP Record Route Option*, but they do *not* install any reservation state or any routing state in the routers. Each BGP router in the transit AS's must insert the associated AS number (and its own IP address) in the Probe messages. The BR's send rejection messages (Error) back to the sources if the destination is unreachable or a reservation loop is detected.

A BR at the destination domain may receive Probe messages from multiple reservation sources. It uses the information carried in the Probe messages to construct an AS-level graph from which a loop-free sink tree is formed.

Reservation sinks return Graft messages to set up the appropriate reservations inside the network. The Graft message uses the previously collected routing information and traverses exactly the reverse path that the Probe message took. Among other things, each Graft message consists of a *sink-tree ID*. A sink-tree ID is a network-wide unique number that is generated by the sink border router. For example, a sink-tree ID can be the combination of the reservation destination AS number and the host IP address of the border route that terminates the original Probe message.

When processing a Graft message, each border router interfaces with intra-domain traffic-engineering protocols (such as RSVP-TE) to set up transit reservations within its domain. The transit BR's aggregate the reservations going to the same reservation domain,

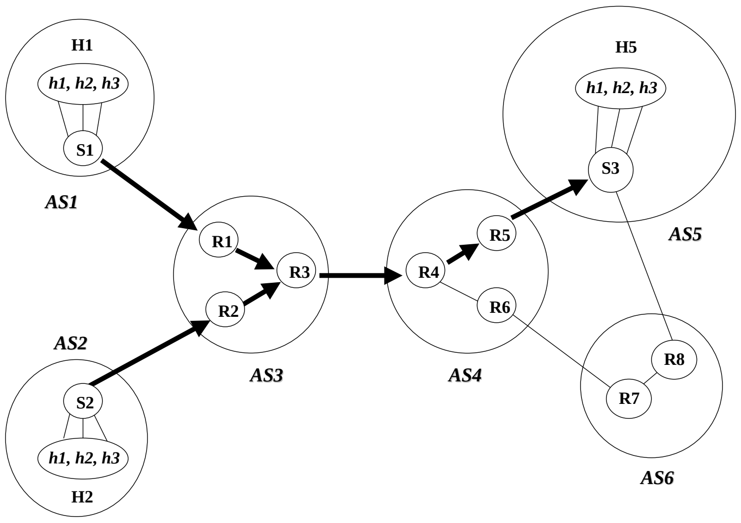


Figure 7-3: Example of a sink tree rooted at S_3

and keep only one reservation entry for each sink tree. In case of admission control failure, the BR's inform the sink by sending Error messages.

Reservation sources and sinks transmit Probe and Graft messages only once during the lifetime of a reservation. BGRP reservations are maintained as "soft state". The border routers (BR's) must periodically exchange Refresh messages with their neighboring BR's to keep their reservations "alive". If a BR does not hear an expected Refresh from a neighboring BR, it assumes that the neighbor is no longer functional. The BR removes all associated reservations. The BGRP protocol also includes Tear messages that BR's may send to remove reservations in neighboring BR's more quickly. In case of route changes, BGP notifies BGRP to re-adjust reservations. To reduce the effect of routing flapping, it may require some dampening mechanism, such as the one being used in BGP [18].

7.4.3 Operation Illustration

Consider a network shown in Figure 7-3. H_1 in AS_1 and H_2 in AS_2 need to set up inter-domain reservations to H_5 in AS_5.

Path Discovery

Initially, S_1 sends a Probe message containing the source ID S_1, the destination ID h_5 ($h_5 \subset \mathcal{H}_5$), the bandwidth requirement $BW_{1,3}$, and an empty route record field.

In this example, S_1 launches a Probe at the behest of a particular host. In a VPN application, however, S_1 could initiate a Probe message toward S_3 directly, in an effort to set up a virtual “trunk” between two domains.

When the Probe message arrives at R_1, the router consults the bilateral agreement between AS_1 and AS_3 and AS_3’s internal network resources. If AS_3 can accept the requested reservation, R_1 inserts its own IP address into the route record field and forwards the Probe message downstream. Otherwise, R_1 sends an Error message back to S_1. The selection of the downstream border router depends on intra-domain traffic engineering constraints and routing policy. In this example, R_1 forwards the message to R_3, as determined from the BGP Next-Hop path attribute. To prevent loops, each router checks whether the current route record already contains the router’s own address.

Assume that R_3, R_4 and R_5 all accept the reservation. When the Probe message arrives at S_3, S_3 determines that the destination ID h_5 belongs to $\mathcal{H}_5$, and thus terminates the probing process. The final route record in the Probe message is (R_1, R_3, R_4, R_5).

Reservation Aggregation

S_3 sends a Graft message back toward S_1 to set up the desired reservation along the path. The Graft message is source-routed using information gathered in the route record of the Probe message. The Graft message contains the bandwidth requirement $BW_{1,3}$, route record (R_1, R_3, R_4, R_5), and a sink-tree ID L. The sink-tree ID uniquely identifies a reservation tree. There may be multiple reservation trees rooted at S_3 that shares the same sink-tree ID. The Graft message traverses exactly the reverse route as listed in the route record. Assume the reservations are successfully made at R_5, R_4, R_3, R_1 and $S1$.

Suppose that S_3 receives another Probe that requests bandwidth $BW_{2,3}$ from S_2 to $\mathcal{H}_5$. S_3 sends back a Graft message to R_5 containing bandwidth requirement $BW_{2,3}$, route record (R_2, R_3, R_4, R_5), and the same sink-tree ID L used in the previous Graft message.

R_5 recognizes this as an increment to the existing sink tree L and increases the reserved bandwidth between R_5 and S_3 to $BW_{1,3} + BW_{2,3}$. Then R_5 forwards the Graft message to R_4, while using the intra-domain reservation protocols of AS_4 to update the internal reservation between R_4 and R_5. The reserved bandwidth between R_4 and R_5 increases to $BW_{1,3} + BW_{2,3}$.

Similarly, router R_4 forwards the Graft message to R_3 while incrementing the reservation between them. When the Graft arrives at R_3, that router creates a new reservation tree branch to R_2 with bandwidth $BW_{2,3}$. The reservation finishes when the Graft arrives at S_2.

If any router R_i cannot set up the new reservation, it sends an Error message back to the sink to inform it of the failure. Along the way, the Error message removes the reservation.

Reservation Management

BGRP reduces the number of reservation entries at routers. For example, in steady state, router R_3 has the following state: sink-tree ID L, adjacent downstream border router R_4, bandwidth reserved to R_4, $BW_{1,3} + BW_{2,3}$, adjacent upstream border routers R_1 and R_2, and bandwidth reserved from each adjacent upstream border router, $BW_{1,3}$ from R_1 and $BW_{2,3}$ from R_2. That is, though there are two reservations being made, R_3 keeps track of only one reservation entry.

R_1, R_2 and R_4 send Refresh messages to R_3 periodically. When R_3 does not receive a Refresh message from one of them within a set period of time, it will free all associated reservations to that neighbor.

7.4.4 Comparing BGRP with RSVP

The BGRP protocol differs from RSVP in three important ways: stateless probing, reservation aggregation, and bundled refresh.

The RSVP PATH message installs routing state at intermediate routers, to guide the RSVP RESV message back to the data sender. Routers must therefore keep both sender and destination information. In a full-meshed network with N nodes, this requires $O(N^2)$ entries. BGRP's Probe messages, however, install no state in routers. BGRP's Graft messages *do* store reservation information, but only $O(N)$ entries, because this information is per-sink, not per-source.

The second difference concerns the combining of reservations. RSVP can combine reservations in two ways. First, RSVP allows multicast receivers to merge their reservations for the same sender (or set of senders) into a single reservation whose size is roughly the *maximum* of the individual reservations. Second, RSVP offers a shared reservation style, where multiple senders take turns sharing a single reservation [71]. BGRP reservation aggregation is different from both of these. BGRP aggregates reservations from different senders to the same receiver by algebraically *adding* them together.

The final difference is that RSVP transmits PATH and RESV messages periodically to refresh each individual reservation separately, while BGRP bundles all reservation messages into one periodic refresh[2].

7.5 BGRP Enhancements

The basic BGRP protocol aggregates reservations into trees, thereby reducing the number of reservations. Reducing the number of reservations obviously shrinks the memory needed to store the control state information. It also reduces the overhead associated with Refresh messages for all these pieces of control state; refresh costs include CPU processing and link bandwidth. These savings take us much of the way toward our goal. However, BGRP sends at least one Probe message and one Graft message between leaf and root for each new inter-domain reservation. Since these messages consume processing CPU and bandwidth, one would like to reducing the control message volume, and thereby add another dimension of scalability to BGRP. This can be done by making the following enhancements to the protocol.

[2]As pointed out in Chapter 3, similar enhancements have recently been proposed for RSVP itself [80].

7.5.1 Over-reservation, Quantization and Hysteresis

Leaf nodes (or reservation senders), in their Probe messages, can request more bandwidth between themselves and the tree root than is currently required. One can think of this as aggregated advance reservations on behalf of unknown parties. Nodes can also coarsely quantize the requested bandwidth, for example, restrict it to multiples of some quantum Q. Hysteresis can also be employed; *e.g.*, if the bandwidth requested by a leaf node has just jumped from $3Q$ to $4Q$ because its bandwidth requirement has just exceeded $3Q$, then that leaf node should not reduce its request back to $3Q$ until its bandwidth requirement drops below some threshold $T < 3Q$.

These changes can dramatically reduce the volume of control messages, as we will quantify in Section 7.6.3.

7.5.2 CIDR Labeling and Quiet Grafting

Suppose that the branches of a BGRP sink tree are labelled with the CIDR prefix[3] associated with the tree root. Then a router on that sink tree will be able to recognize whenever a Probe message that "belongs" to that tree arrives (*i.e.*, the reservation destination belongs to the same CIDR prefix). Also, suppose that this tree node can over-reserve bandwidth between itself and the tree root. These two modifications enable a new tree operation called *quiet grafting*, whereby a new branch can be grafted onto an existing reservation sink tree without any Probe or Graft messages being passed between the grafting node and the tree root.

To demonstrate quiet grafting, let us construct the sink tree shown in Figure 7-3. Suppose that initially S_1 requests 10 units of reserved bandwidth to H_5 in a Probe message. Knowing that H_5 is a popular destination, R_3 inflates this request to 15 units as it processes the passing Probe. When the Graft message returns from tree root S_3, it reserves 15 units at each BR-hop, until reaching R_3. Node R_3 deflates the amount to 10 units and passes the Graft back toward R_1 and S_1. Now 10 units of bandwidth have been reserved from S_1 to

[3]CIDR (Classless Inter-Domain Routing) [15, 16] advertises each route prefix together with its mask and does not depend on the classical Class A, B and C addressing. It allows the routers to aggregate the routes.

R_3, and 15 units have been reserved from R_3 to S_3. In R_3's own internal bookkeeping for these 15 units, R_3 considers 10 units as "belonging" to R_1's tree branch, and it considers 5 units as over-reserved. Now suppose S_2 requests 3 units of reserved bandwidth to H_5 in its Probe message. When this Probe reaches R_3, R_3 recognizes that it is already placed on a sink tree to H_5 and that there is sufficient excess bandwidth already reserved between R_3 and S_3 to satisfy S_2's needs, so that R_3 can handle the new request directly itself, without propagating the Probe further downstream. Therefore, R_3 terminates the Probe, adjusts its internal bookkeeping to assign 3 of its 5 excess bandwidth units to this new tree branch, and launches a Graft message back toward R_2 and S_2. This Graft establishes a 3-unit reservation between leaf S_2 and grafting node R_3.

7.5.3 Self-Healing

When a route changes, BGRP has the option of moving the affected reservations to the new route, without demolishing the entire reservation tree and re-creating the tree from scratch. Assume that the reservation tree is labeled with the destination CIDR prefixes, as described above. When a tree node detects a route change, it can initiate a new Probe toward the sink. When this Probe reaches a downstream router on the stable part of the old reservation tree, that router can respond with a Graft and thus repair the part of the reservation between the two routers. We call this process *self-healing*.

7.5.4 Reservation Damping

Labovitz *et al.*[20, 131] have shown that, of three million BGP route changes each day, 99% of the changes did not reflect real network topological changes. If routers make BGRP reservation changes in response to every route change, there could be a high volume of nearly worthless reservation messages in the network. On the other hand, if the routers do not move a reservation, and the route change turns out to be legitimate and stable, then the data will have lost its reservation. This is the trade-off in deciding when to adjust reservations.

Here, we propose a damping function for BGRP. The goal of reservation damping is to delay the initiation of the self-healing process until the changing routes have stabilized. The delay depends on the probability of future instability of the route. Routes that change frequently will be delayed longer. Similar to the staged refresh technique that we have proposed in Chapter 3, we propose an exponential function $\tau = (1+\Delta)^n \cdot T$ for computing the delay τ between Probes sent due to route changes. Δ and T are the parameters to adjust the damping, and n is the number of route changes measured in a time interval.

7.6 Protocol Scaling Evaluation

7.6.1 Topological Distribution of Demand

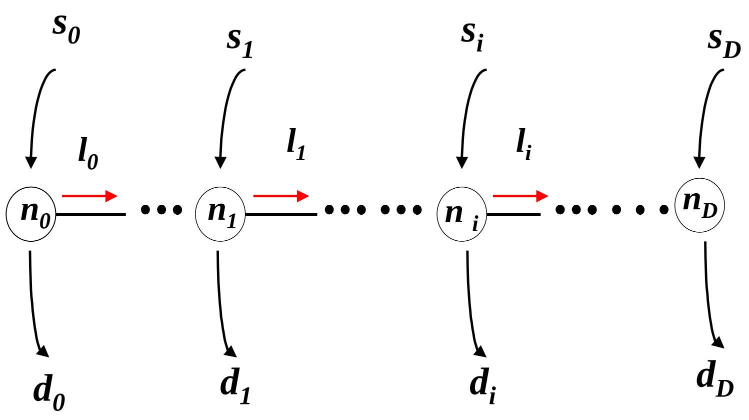


Figure 7-4: Model for analyzing the topological distribution of demand

We use the simple model in Fig. 7-4 to compare the scaling properties of BGRP and RSVP. The model depicts a progression of domains along an Internet path, with access networks toward the left and right and backbone networks near the middle of the topology. Traffic flows from left to right only. We define D as the maximum edge-to-edge distance, measured as the number of AS. A “node” n_i in the figure represents an inter-domain traffic exchange point, which can be either a point of presence (POP) or a NAP. (In the real network from which this model is abstracted, each node n_i could actually contain many routers and interconnect many AS.) A “link” l_i in the figure represents an aggregation of all the real links that transport traffic from domain i to domain $i + 1$. The model also includes a reverse-directed link from $i+1$ to i, which is not shown in the figure. In addition to the diameter D, our model is characterized by the quantities s_i and d_i: s_i is the number of inter-domain reservation sources coming into n_i, and d_i is the number of reservations sinks reached through n_i, not including those that n_i reaches via l_i.

In this model, the number of RSVP flows (i.e., source-destination pairs) on the uni-directional link l_i is given by

$$L_i^{\text{RSVP}} = \sum_{j=0}^{i} s_j \sum_{k=i+1}^{D} d_k$$

and the number of BGRP flows (i.e., one per destination) on l_i is given by

$$L_i^{\text{BGRP}} = \sum_{k=i+1}^{D} d_k.$$

Node n_i handles traffic in both directions, so the number of reservation flows traversing n_i is given by

$$N_i^{\text{RSVP}} = \sum_{j=0}^{i} s_j \sum_{k=i}^{D} d_k + \sum_{j=i}^{D} s_j \sum_{k=0}^{i} d_k - s_i d_i$$

for RSVP and

$$N_i^{\text{BGRP}} = \sum_{j=0}^{D} d_j$$

for BGRP.

We now study both RSVP and BGRP in different networking scenarios, computing the number of flows and associated gains. We set $D = 9$, which is the longest possible AS path length in the Internet today [55]. We simulated models with a total of 100 source and sink border routers. We assume that every source border router desires to set up a reservation to every sink border router.

Case 1: Flat Topology: The number of reservation sources and sinks are identical at each n_i; i.e., $s_i = d_i = 5$, for $0 \leq i \leq 9$.

Case 2: Hierarchical Topology: We set $s_i = d_i = 11$, for $3 \leq i \leq 6$, and $s_i = d_i = 1$, for all other i. This models a hierarchical network where stub domains only contribute a small portion of the overall reservation flows, while most of the reservations are present at a few core transit domains.

Case 3: Selected Source: We set all s_i to 1, and all d_i to 9. This reflects a network where the number of data sources is small, but the number of sinks (data receivers) is large. This is a typical scenario for web applications.

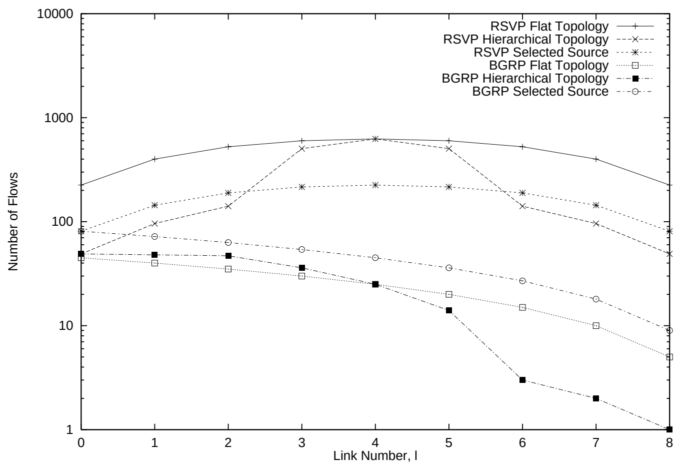
The results are shown in Fig. 7-5. Not surprisingly, BGRP maintains fewer reservations than RSVP. Fig. 7-5(c) shows that the largest gain occurs in the center of the network. Also, Fig. 7-5(b) shows that for RSVP, the number of reservations at a node depends on the node's topological location, whereas for BGRP, every node has the same number of reservations.

7.6.2 Reservation Dynamics

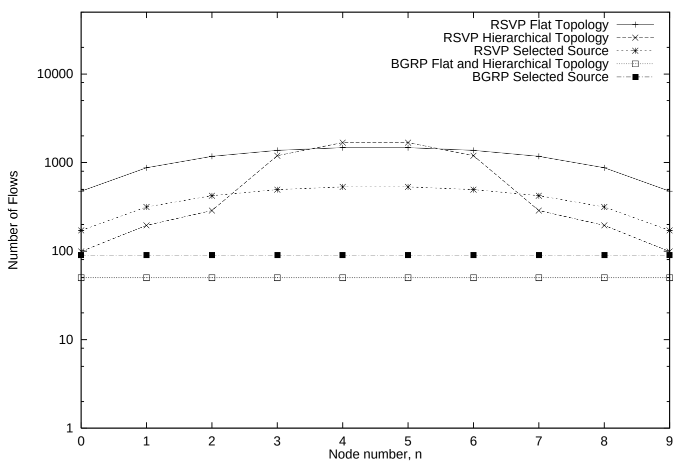
We now consider the effect of reservation dynamics on our performance analysis. For both RSVP and BGRP, we shall determine the control state overhead and the control message overhead as functions of the arrival rate of individual flows, the mean lifetime of a flow, and the protocol refresh interval. We assume that reservations are explicitly torn down when no longer needed, and that the blocking rate for reservations is negligible for our purposes.

Recall that the "virtual hop" between two "adjacent" border routers is called a BR-hop. We define the sequence of border routers visited by an end-to-end traffic flow as an edge-to-edge BR-path. For RSVP, each edge-to-edge BR-path establishes its own reservation. Let F be the total number of edge-to-edge BR-paths crossing a given border router R_i. For BGRP, these paths are aggregated into trees. Let T be the total number of sink trees formed by the paths through R_i. To keep this analysis simple, we assume that each sink tree at R_i is aggregating the same number $f = F/T$ of edge-to-edge BR-paths.

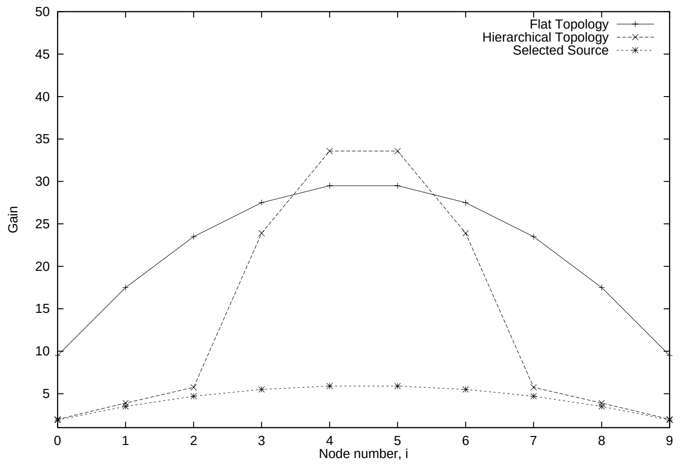
Now let us model the end-to-end flows desiring reservations. Any number of end-to-end reserved flows may be multiplexed onto a given edge-to-edge BR-path. We assume that the end-to-end reserved flows for one path arrive in a Poisson stream of rate λ, and that the flow reservation lifetimes are exponentially distributed with mean $1/\mu$. Then the



(a) Number of reservations per link



(b) Number of reservations per node



(c) The gain, $N_i^{\text{RSVP}}/N_i^{\text{BGRP}}$

Figure 7-5: Worst case scaling comparison between RSVP and BGRP.

number of reserved end-to-end flows on one edge-to-edge BR-path is Poisson distributed with mean $\rho = \lambda/\mu$, and the number of reserved end-to-end flows on any given sink tree is Poisson distributed, with mean $\rho \cdot f$.

Let us determine the state counts of the protocols. In case of RSVP, the probability that the edge-to-edge BR-path has a reservation, *i.e.*, that at least one end-to-end flow is reserved, is $1 - e^{-\rho}$. Therefore, the number of reserved edge-to-edge BR-paths at R_i is binomially distributed with mean $(1 - e^{-\rho}) \cdot F$. This is the average amount of RSVP control state for R_i. For large ρ, this approaches F, while for small ρ, this approaches 0. Now let us determine the BGRP state count at R_i. The probability that a given sink tree has a reservation at R_i, i.e., that at least one end-to-end flow on at least one edge-to-edge BR-path on that tree has requested a reservation, is $1 - e^{-\rho \cdot f}$. The number of reserved trees on R_i is thus binomially distributed with mean $(1 - e^{-\rho \cdot f}) \cdot T = (1 - e^{-\rho \cdot F/T}) \cdot T$. This is the average amount of BGRP control state for R_i. For large ρ, this approaches T, while for small ρ, this approaches 0. We conclude that the state advantage of BGRP with respect to RSVP is more pronounced when ρ is large, where this gain approaches F/T. Figure 7-6 shows the mean number of simultaneous reservations for BGRP and RSVP as ρ ranges from 0.001 to 10; here we assume $F = 100,000$ and $T = 1,000$. For instance, when $\rho = 1$, the gain is 63. When $\rho = 10$, the gain has essentially reached its maximum value of 100.

Next we will compare the message rates for the two protocols. We will tally the control messages associated with reservations on one given unidirectional BR-hop. (Note that some of these "associated" messages actually travel on the reverse BR-hop.) When a new end-to-end flow for the given BR-hop is born, this counts as two messages in either protocol: PATH and RESV for RSVP, or Probe and Graft for BGRP. Removing the reservation takes a single TEAR message, in either protocol. Since refresh is bidirectional, in both protocols, we count the refreshing of one reservation on a given BR-hop as two units of control message processing. (Refreshes for multiple reservations on the same BR-hop are assumed to be *processed* separately, even though they may be *transmitted* in a bundle.) Let η be the refresh rate. For RSVP, the average message rate for each edge-to-edge BR-path on the given BR-hop is $3\lambda + 2\eta \cdot (1 - e^{-\rho})$. Hence, the average RSVP message rate for the

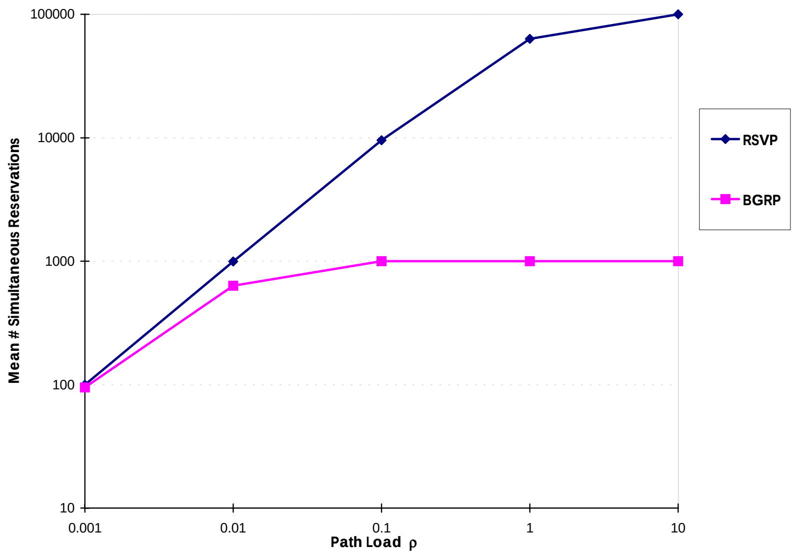


Figure 7-6: The number of BGRP and RSVP reservations as functions of load ρ

BR-hop is $[3\lambda + 2\eta \cdot (1 - e^{-\rho})] \cdot F$. For BGRP, the average message rate for one sink tree on the given BR-hop is $3\lambda \cdot f + 2\eta \cdot (1 - e^{-\rho \cdot f})$. Hence, the average BGRP message rate for the BR-hop is $[3\lambda \cdot f + 2\eta \cdot (1 - e^{-\rho \cdot f})] \cdot T$, which equals $3\lambda \cdot F + 2\eta \cdot T \cdot (1 - e^{-\rho \cdot F/T})$. If λ is much larger than η, then BGRP's Probe, Graft and Tear activities dominate its Refreshes, and RSVP's initial PATH and RESV messages dominate their refreshed versions. In this case, RSVP and BGRP have the same message rates. (Fortunately, the over-reservation techniques that we proposed in Section7.5 can dramatically reduce BGRP's message processing overhead at this end of the spectrum; we will analyze these improvements shortly in Sec. 7.6.3 below.) On the other hand, if η is much larger than λ, so that Refresh activity dominates, and if ρ is large, then BGRP does better than RSVP by a factor of F/T. Figure 7-7 shows the average message rates per second, for BGRP and RSVP, as η ranges from 0.0003 to 3.0 refreshes per second; here we assume $\lambda = 0.001$ flows per second, $F = 100,000, T = 1,000$, and $\rho = 10$. For instance, when $\eta = 0.03$ (i.e., when the refresh interval is about 30 seconds), then the gain (i.e., the ratio of RSVP messages to BGRP

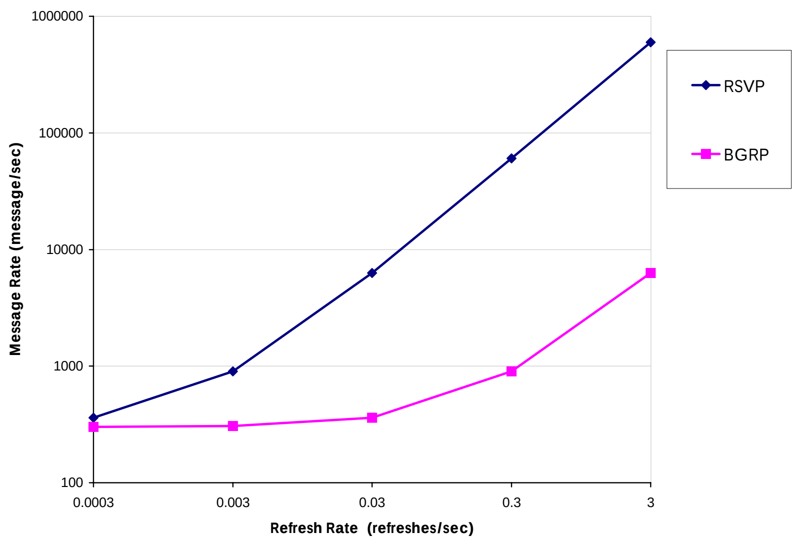


Figure 7-7: BGRP and RSVP message rates as functions of refresh rate η

messages) is 18. When $\eta = 3.0$, the gain is 95.8.

7.6.3 Over-reservation, Quantization and Hysteresis

We showed in Section 7.6.2 that BGRP has fewer messages to process than RSVP, provided that most messages are Refreshes rather than Probes, Grafts, and Tears. In this section we show how hysteresis can be used to curb the Probe, Graft and Tear activity. The savings in message processing come at the cost of some wasted bandwidth, because the aggregate reservation sometimes exceeds the sum of the component flow requests.

We shall model a single aggregate reservation on a single BR-hop. We assume that the blocking rate for reservations is extremely low, negligible for the purpose of measuring mean message rates. Any number of end-to-end flow reservations can be multiplexed into this aggregate. Assume that these end-to-end reserved flows arrive in a Poisson stream of rate λ, and that the flow reservation lifetimes are exponentially distributed with mean $1/\mu$. Then the number of reserved end-to-end flows in our aggregate reservation is Poisson

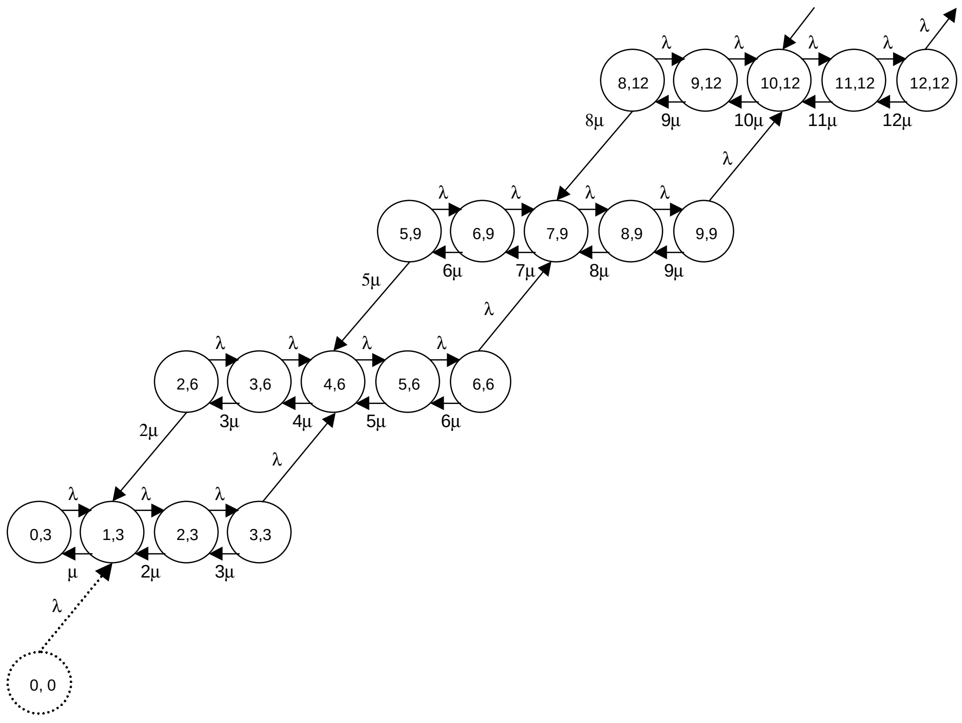


Figure 7-8: State transition diagram for $Q = 3$.

distributed with mean $\rho = \lambda/\mu$.

Assume that each end-to-end flow requires one unit of bandwidth. We constrain the aggregate reservation always to be a multiple of some quantum size $Q \geq 2$. Whenever the aggregate reservation is kQ units, and this is just barely enough to satisfy the current individual flows, and a new end-to-end flow reservation is requested, then the aggregate reservation jumps to $(k+1)Q$. This new quantum will only be relinquished when the sum of the individual flow requests drops to $(k-1)Q+1$. The amount of bandwidth wasted due to over-reservation by this technique is less than $2Q$ units.

We can model this system as a two-dimensional Markov process with state (x, y), where x is the number of currently reserved end-to-end flows, and y is the current aggregate reservation. Figure 7-8 shows the state transition diagram for $Q = 3$. The only valid states are the following: state (0,0), which we will ignore because it is transient; states (x, Q), where $0 \leq x \leq Q$; and states $(x, k \cdot Q)$, for $k \geq 2$, where $(k-2) \cdot Q + 2 \leq x \leq k \cdot Q$. The state transition diagram has a structure that makes it straightforward to solve for the

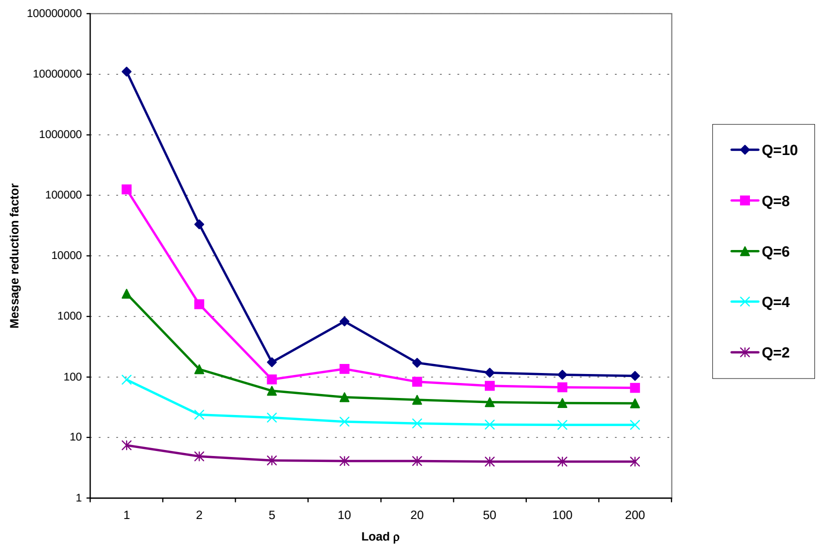


Figure 7-9: BGRP message reduction factor as function of Q and ρ.

steady-state probabilities. Note that, for most values of x, there are two possible values of y, i.e., two states. However, certain special values of x have only one possible y value, i.e., one state. These special values of x are: $x = 0$, for which y must equal Q, and $x = Q+1, 2Q+1, 3Q+1, ...$, for which the respective values of y must be $2Q, 3Q, 4Q, ...$. Therefore, at these special points, the joint probability distribution $\pi(x, y)$ matches the marginal distribution of x by itself, which we already said is Poisson. It is straightforward to determine the probabilities of all the other states in terms of these special states, thereby completing the analysis. For all $k \geq 1$:

$$\pi(0, Q) = e^{-\rho}$$

$$\pi(((k-1)Q+1), kQ) = \frac{e^{-\rho} \cdot \rho^{((k-1)\cdot Q+1)}}{((k-1)\cdot Q+1)!}$$

For $(k-1)Q+2 \leq x \leq kQ$:

$$\pi(x, k \cdot Q) = \frac{e^{-\rho} \cdot \rho^x \cdot \sum_{i=0}^{k \cdot Q - x}[\rho^i \cdot (k \cdot Q - i)!]}{x! \cdot \sum_{i=0}^{Q-1}[\rho^i (k \cdot Q - i)!]}$$

$$\pi(x, (k+1)Q) = \frac{e^{-\rho} \cdot \rho^x \cdot \sum_{i=k \cdot Q+1-x}^{Q-1} [\rho^i \cdot (k \cdot Q - i)!]}{x! \cdot \sum_{i=0}^{Q-1}[\rho^i \cdot (k \cdot Q - i)!]}$$

The rate of BGRP control messages (Probes + Grafts + Tears) is:

$$3\lambda \cdot e^{-\rho} \cdot \sum_{k=1}^{\infty} \left(\frac{\rho^{(k \cdot Q)}}{\sum_{i=0}^{Q-1}[\rho^i \cdot (k \cdot Q - i)!]} \right)$$

This compares very favorably to the message rate 3λ without hysteresis. Figure 7-9 shows the message rate reduction factor for various values of Q and ρ. For example, if reservations are quantized in blocks of 10, and $\rho = 100$, then quantization and hysteresis reduce the BGRP message rate by a factor of about 100. (The curves in Fig. 7-9, while roughly decreasing, are not strictly monotonic. The ripples in the plots are due to the many "corners" in the state transition diagram of Fig. 7-8. For a given quantum size Q, as ρ increases, the bulk of the probability mass zigzags upward and to the right through the state space. Since vertical state transitions produce protocol messages while horizontal transitions do not, these zigs and zags around the corners can produce ripples in the messaging efficiency plot.)

The analysis above dealt with a single aggregate reservation on a single BR-hop. It is a good model for a system where the *leaves* of sink trees are the only BR-hops that can initiate an over-reservation. However, if BR-hops *anywhere* on the tree can deliberately over-reserve, and if the quiet grafting described in Section 7.5 is done, then additional savings in message rate are possible.

Note that, in order for over-reservation to be helpful, the over-reserving router must recognize the future traffic on whose behalf the over-registration was made, without having to send a new Probe message. For the tree leaves that over-reserve, simple caching of the destination network IDs associated with each tree would work. For more complex situations, see the labeling discussion on quiet grafting in Section 7.5.

7.7 Related Work

Recently, several authors have addressed scalable resource reservation, using either a server-based or a router-based approach.

In server-based approaches, each domain has a bandwidth broker (or agent) which is responsible for selecting and setting up the aggregated reservation sessions. This approach has the advantage of removing the message processing and storage burden from routers. However, synchronizing reservation information among the bandwidth brokers and the routers may be complex. No aggregation takes place, so that each broker still has to deal with the requests of individual flows. Also, care has to be taken so that the broker does not become a single point of failure for the domain. Variations of the server-based approach have been described by Blake *et al.* [59], Schelen and Pink [132], Berson *et al.* [133], and Terzis *et al.* [134]. The latter proposal suggests a two-tier system where, within each domain, hosts use intra-domain reservation protocols such as RSVP to set up reserved flows. Inter-domain reservation protocols set up coarsely-measured reserved flows between domains. However, the proposal leaves the actual mechanism undefined.

Awduche *et al.* [97], Guérin *et al.* [135] and Baker *et al.* [136] have proposed a router-based approach by modifying RSVP to support scalable reservation. (Awduche's LSP tunnels [97] are designed to support *intra*-domain traffic engineering, but may also be used to set up trunks crossing multiple domains.) These proposals aggregate per-application reservation requests into reservation "trunks" between pairs of domains, by modifying sender template and session objects in RSVP to carry address and mask ("CIDR blocks") or autonomous system (AS) numbers instead of 5-tuples (sender IP address, sender port, receiver IP address, receiver port, protocol). However, this implies that routers in the backbone may have to maintain reservation state proportional to the square of the number of CIDR blocks or autonomous systems in full-meshed backbone networks.

7.8 Conclusion

Resource reservation must operate in an efficient and scalable fashion, to accommodate the rapid growth of the Internet. In this chapter, we presented an inter-domain reservation protocol, called the Border Gateway Reservation Protocol (BGRP), in which reservations are aggregated along sink trees.

Each provider's domain may use its own method to manage network resources. BGRP builds a sink tree for each of the stub domains. Each sink tree aggregates bandwidth reservations from all data sources in the network. Since backbone routers maintain only the sink tree information, the total number of reservations at each router scales linearly with the number of Internet domains. BGRP maintains these aggregated reservations using "soft state."

BGRP scales well in terms of control state, message processing, and bandwidth. It reduces control state by aggregating reservations; this reduces their number, and thereby reduces the memory needed to store the control information.

Control message processing is the most important scalability issue. The cost of processing reservation messages depends on the volume of requests for new reservations, the volume of existing reservations requiring periodic refreshing, and the refresh frequency. BGRP economizes on all of these components. First, when we allow routers to over-reserve bandwidth with BGRP, then small reservations can join and leave the reservation sink tree without disturbing the entire tree. This reduces the volume of reservation set-up messages. Second, by aggregating reservations, BGRP reduces their number, and this proportionally reduces the refresh processing burden. Third, BGRP needs less frequent refreshes than does RSVP, for the following reason. RSVP control messages are unreliable and thus must be repeated at about three times the state-timeout interval, while BGRP refresh messages are transferred reliably hop-by-hop. The mechanisms described above reduce the number of control messages. Not only does this reduce the burden on the routers to process these messages, it also reduces the link bandwidth cost to transmit these messages.

Chapter 8

Summary and Future Work

We conclude this dissertation with a discussion of the merits of our various reservation signaling proposals, a short list of outstanding problems in the area of Internet resource management, and a summary of our main contributions.

8.1 Summary of the Proposals

In Chapter 2, we investigated the necessity of having resource reservation in the Internet, and the scalability issues involved in reservation signaling design and implementation.

Chapter 3 described a new mechanism, *staged refresh timer*, to improve RSVP's scalability by reducing the soft-state refresh overhead, while enhancing the reliability in message delivery. We have shown that the mechanism was very effective when adapted by all network routers.

In Chapter 4 showed that with the current growth rate, the Internet cannot operate with one signaling protocol to setup and manage all the reservations. In addition, most of existing protocols have the potentially state "explosion" problem when it comes to handle reservation states. Thus, we proposed a *hierarchical reservation model*, where end-users have the option to signal for reservations at the application layer, and the network providers always interconnect each other with large and shared reservation trunks. At network border, application-layer reservations are aggregated into the provider-layer reserved trunks.

In Chapter 6, we showed the design and implementation of a lightweight application-layer reservation protocol, YESSIR, that supports real-time streaming applications that use the RTP protocol. It has incorporated some of the features studied in Chapter 5 such as partial reservation, fast resource retry and one-pass reservation model.

We have described the design and performance analysis of an inter-domain reservation protocol, BGRP, in Chapter 7. BGRP interfaces with BGP to distribute and maintain control states (such as QoS and MPLS label) throughout the Internet in a scalable fashion.

8.2 Our Main Contributions

The main contributions of this dissertation are the following:

- There is a *strong* need for resource reservation in the Internet, especially at network interconnection links.

- The Internet reservation scalability problems can be solved with the hierarchical reservation model, that consists of an application-layer reservation component and a provider-level reservation component.

- With careful protocol design, we can develop very efficient application-layer reservation protocols, that have the features such as fast to setup, fast to recover from admission control failure and simple to process.

- By applying reservation aggregation can reduce the total number of reservations in the Internet. At provider-level, sink-tree aggregation technique can reduce provider-level reservations to $O(N)$, where N is the total number of AS's.

Bibliography

[1] Internet Software Consortium, “Internet domain survey.” http://www.isc.org/ds/.

[2] Telstra Network, “Telstra internet network performance reports.” http://www.telstra.net/ops/.

[3] T. Bates, “The CIDR report.” http://www.employees.org/˜tbates/cidr-report.html.

[4] A. Odlyzko, “The Internet and other networks: Utilization rates and their implications,” technical report, AT&T Labs, Florham Park, NJ, Feb. 2000.

[5] R. E. Kahn and V. G. Cerf, “What is the internet (and what makes it work),” tech. rep., CNRI, MCI Worldcom, Dec. 1999.

[6] UUnet, “Our network.” http://www.uu.net/network/.

[7] EP.net, “Exchange point information.” http://www.ep.net/.

[8] W. B. Norton, “Internet service providers and peering,” in *Proc. of NANOG 19*, (Albuquerque, New Mexico), June 2000.

[9] K. G. Coffman and A. M. Odlyzko, “The size and growth rate of the internet,” tech. rep., AT&T Labs - Research, Florham Park, New Jersey, July 1998.

[10] G. Malkin, “RIP version 2,” RFC 2453, Internet Engineering Task Force, Nov. 1998.

[11] J. Moy, “OSPF version 2,” RFC 2178, Internet Engineering Task Force, July 1997.

[12] ISO (International Organization for Standardization), "Information processing systems - data communications - intermediate system to intermediate system intra-domain routing protocol," ISO Standard ISO 10589, International Organization for Standardization, Geneva, Switzerland, 1992.

[13] D. Katz, "OSPF and IS-IS - a comparative anatomy," in *NANOG Spring Meeting*, (Albuquerque, New Mexico), June 2000.

[14] Y. Rekhter and T. Li, "A border gateway protocol 4 (BGP-4)," Internet Draft, Internet Engineering Task Force, Nov. 2001. Work in progress.

[15] Y. Rekhter and T. Li, "An architecture for IP address allocation with CIDR," RFC 1518, Internet Engineering Task Force, Sept. 1993.

[16] V. Fuller, T. Li, J. Yu, and K. Varadhan, "Classless inter-domain routing (CIDR): an address assignment and aggregation strategy," RFC 1519, Internet Engineering Task Force, Sept. 1993.

[17] P. Traina, "Autonomous system confederations for BGP," RFC 1965, Internet Engineering Task Force, June 1996.

[18] C. Villamizar, R. Chandra, and R. Govindan, "BGP route flap damping," RFC 2439, Internet Engineering Task Force, Nov. 1998.

[19] T. Bates, R. Chandra, and E. Chen, "BGP route reflection - an alternative to full mesh IBGP," RFC 2796, Internet Engineering Task Force, Apr. 2000.

[20] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet routing instability," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Cannes, France), Sept. 1997.

[21] C. Labovitz, A. Ahuja, A. Abose, and F. Jahanian, "An experimental study of delayed internet routing convergence," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Stockholm, Sweden), Aug. 2000.

[22] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, “RSVP: a new resource ReSerVation protocol,” *IEEE Network*, vol. 7, pp. 8–18, Sept. 1993.

[23] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, and S. Jamin, “Resource ReSerVation protocol (RSVP) – version 1 functional specification,” RFC 2205, Internet Engineering Task Force, Sept. 1997.

[24] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: a transport protocol for real-time applications,” RFC 1889, Internet Engineering Task Force, Jan. 1996.

[25] L. Zhang, S. Shenker, D. Clark, C. Huitema, S. Deering, and D. Ferrari, “Reservations or no reservations,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Boston, Massachusetts), Apr. 1995. panel-discussion slides.

[26] F. Baker, J. Crowcroft, R. Guerin, H. Schulzrinne, and L. Zhang, “Reservations about reservations,” in *Proc. of IFIP Fifth International Workshop on Quality of Service (IWQOS ’97)*, (New York, NY), May 1997.

[27] H. Schulzrinne, “Guide to nevot 3.34.” http://www.cs.columbia.edu/˜hgs/nevot/.

[28] UCL Multimedia, “Robust audio tool (RAT).” http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/.

[29] UCB/LBNL, “vat – LBNL audio conferencing tool.” http://www-nrg.ee.lbl.gov/vat/.

[30] R. Braden, D. Clark, and S. Shenker, “Integrated services in the internet architecture: an overview,” RFC 1633, Internet Engineering Task Force, June 1994.

[31] J. Wroclawski, “The use of RSVP with IETF integrated services,” RFC 2210, Internet Engineering Task Force, Sept. 1997.

[32] J. Wroclawski, “Specification of the controlled-load network element service,” RFC 2211, Internet Engineering Task Force, Sept. 1997.

[33] S. Shenker, C. Partridge, and R. Guerin, “Specification of guaranteed quality of service,” RFC 2212, Internet Engineering Task Force, Sept. 1997.

[34] S. Shenker, "Fundamental design issues for the future internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, Sept. 1995.

[35] L. Breslau and S. Shenker, "Best-effort versus reservations: A simple comparative analysis," *ACM Computer Communication Review*, vol. 28, pp. 3–16, Sept. 1998.

[36] UUnet, "Latency statistics." http://www.uu.net/network/latency/index.html.

[37] Sprint, "Sprint network latency statistics." http://www.sprintbiz.com/internet_solutions/SLAs.htr

[38] V. Paxson, "End-to-end internet packet dynamics," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Cannes, France), Sept. 1997.

[39] V. Paxson, *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California at Berkeley, Berkeley, California, May 1997.

[40] International Telecommunication Union (ITU), "Transmission systems and media, general recommendation on the transmission quality for an entire international telephone connection; one-way transmission time," Recommendation G.114, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Mar. 1993.

[41] International Telecommunication Union, "Dual rate speech coder for multimedia communications transmitting at 5.3 and 6.3 kbit/s," Recommendation G.723.1, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Mar. 1996.

[42] SWITCH, "Switchlan traffic statistics." http://www.switch.ch/lan/stat/.

[43] NORDUnet, "Nordunet network statistics." http://www.nordu.net/stats/.

[44] ABOVE.net, "Above.net's real-time network status." http://stats.sjc.above.net/traffic/.

[45] BBC Internet Services, "Internet link usage." http://support.bbc.co.uk/support/mrtg/internet/.

[46] A. Odlyzko, "The history of communications and its implications for the Internet," technical report, AT&T Labs, Florham Park, NJ, June 2000.

[47] Commission of European Communities, “Commission recommendation on leased lines interconnection pricing in a liberalised telecommunications market,” tech. rep., Commission of European Communities, Nov. 1999.

[48] B. S. Arnaud, “CANARIE – CA*net 3 – the customer empowered networking revolution,” in *Proc. of First Australian Advanced and Innovative Internet Workshop*, (Adelaide, Australia), Mar. 2000.

[49] C. Stefano, “Rural connectivity: Economics,” in *Second Advisory Committee Meeting for National Coordination Office for Information Technology Research and Development*, (Arlington, VA), June 1997.

[50] R. Frieden, “Does a hierarchical internet necessitate multilateral intervention?,” in *Proc. of 28th Research Conference on Communication, Information and Internet Policy*, (Alexandria, Virginia), Sept. 2000.

[51] B. Zhang, “Assessing the WTO agreements on China’s telecommunications regulatory reform and industrial liberalization,” in *Proc. of 28th Research Conference on Communication, Information and Internet Policy*, (Alexandria, Virginia), Sept. 2000.

[52] K. G. Coffman and A. M. Odlyzko, “Internet growth: Is there a “Moore’s Law” for data traffic?,” tech. rep., AT&T Labs - Research, Florham Park, New Jersey, July 2000.

[53] NLANR, “Assessing average hop count of a wide area internet packet.” http://www.nlanr.net/NA/Learn/wingspan.html.

[54] V. Paxson, “End-to-end routing behavior in the internet,” *IEEE/ACM Transactions on Networking*, vol. 5, pp. 601–615, Oct. 1997.

[55] NLANR, “NLANR AS path lengths.” http://moat.nlanr.net/ASPL/.

[56] BELNET, “Belnet: The belgian research network.” http://www.belnet.be/main␣uk.html.

[57] DoIT Network Engineering, “Internet traffic flow size analysis.” http://net.doit.wisc.edu/data/flow/size/.

[58] T. cker Chiueh, A. Neogi, and P. Stirpe, “Performance analysis of an RSVP-capable router,” in *Proc. of 4th Real-Time Technology and Applications Symposium*, (Denver, Colorado), June 1998.

[59] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, “An architecture for differentiated service,” RFC 2475, Internet Engineering Task Force, Dec. 1998.

[60] I. Stoica and H. Zhang, “Providing guaranteed service without per flow management,” *ACM Computer Communication Review*, vol. 29, pp. 81–94, Oct. 1999.

[61] R. Guerin, L. Li, S. Nadas, P. Pan, and V. Peris, “The cost of QoS support in edge devices: An experimental study,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (New York), Mar. 1999.

[62] B. Braden, “ISI RSVP host implementation,” software release, ISI, Marina del Rey, California, June 1997.

[63] P. Pan and R. Guerin, “IBM Research Center RSVP router implementation,” software documentation, IBM T. J. Watson Laboratory, June 1997.

[64] LightReading, “The internet core router test,” in *LightReading Test*, Mar. 2001. http://www.lightreading.com/testing/.

[65] P. P. Pan and H. Schulzrinne, “YESSIR: A simple reservation mechanism for the Internet,” in *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, (Cambridge, England), pp. 141–151, July 1998. also IBM Research Technical Report TC20967.

[66] P. Pan and H. Schulzrinne, “Processing overhead studies in resource reservation protocols,” in *17th International Teletraffic Congress*, (Salvador da Bahia, Brazil), Sept. 2001.

[67] G. Gaines and L. Salgarelli, "RSVP implementation survey," tech. rep., Institute for Information Technology of the National Research Council of Canada, July 1997.

[68] K. Cho, "Managing traffic with ALTQ," in *1999 USENIX Annual Technical Conference,* (Montery, California, USA), June 1999.

[69] M. Karsten, J. Schmitt, and R. Steinmetz, "Implementation and evaluation of the KOM RSVP engine," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Anchorage, Alaska), Apr. 2001.

[70] D. J. Mitzel, D. Estrin, S. Shenker, and L. Zhang, "An architectural comparison of ST-II and RSVP," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Toronto, Canada), June 1994.

[71] D. J. Mitzel and S. Shenker, "Asymptotic resource consumption in multicast reservation styles," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (London, UK), pp. 226–233, Sept. 1994.

[72] A. Birman, V. Firoiu, R. Guérin, and D. Kandlur, "Provisioning of RSVP-based services over a large ATM network," Research Report RC 20250, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, New York, New York, 1995.

[73] S. Shenker and L. Breslau, "Two issues in reservation establishment," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Cambridge, Massachusetts), Sept. 1995.

[74] P. Pan and H. Schulzrinne, "Staged refresh timers for RSVP," in *Proceedings of Global Internet*, (Phoenix, Arizona), Nov. 1997. also IBM Research Technical Report TC20966.

[75] R. A. Guerin, S. Kamat, and S. Herzog, "QoS path management with RSVP," in *Proc. of Global Internet (Globecom)*, (Phoenix, Arizona), Nov. 1997.

[76] M. Karsten, J. Schmitt, L. Wolf, and R. Steinmetz, “An embedded charging approach for RSVP,” in *Proceedings of 6th IEEE/IFIP International Workshop on Quality of Service*, (Napa, California), pp. 91–100, IEEE/IFIP, May 18–20 1998.

[77] M. Yajnik, J. Kurose, and D. Towsley, “Packet loss correlation in the MBone multicast network,” in *Proceedings of Global Internet*, (London, England), Nov. 1996.

[78] S. Casner, H. Schulzrinne, *et al.*, “Frequently asked questions (FAQ) on the multicast backbone (mbone).” http://www.cs.columbia.edu/˜hgs/internet/mbone-faq.html.

[79] L. Mathy, D. Hutchison, and S. Simpson, “Modelling and improving flow establishment in RSVP,” in *Proc. of Protocols for High Speed Networks (PfHSN)*, Aug. 1999.

[80] L. Berger, D. Gan, G. Swallow, P. Pan, F. Tommasi, and S. Molendini, “RSVP refresh overhead reduction extensions,” RFC 2961, Internet Engineering Task Force, Apr. 2001.

[81] G. Varghese and A. Lauck, “Hashed and hierarchical timing wheels: Efficient data structures for implementing a timer facility,” *IEEE/ACM Transactions on Networking*, vol. 5, pp. 824–834, Dec. 1997.

[82] S. Floyd and V. Jacobson, “The synchronization of periodic routing messages,” *IEEE/ACM Transactions on Networking*, vol. 2, pp. 122–136, Apr. 1994.

[83] G. Armitage, “Support for multicast over UNI 3.0/3.1 based ATM networks,” RFC 2022, Internet Engineering Task Force, Nov. 1996.

[84] R. Talpade and M. Ammar, “Multicast server architectures for MARS-based ATM multicasting,” RFC 2149, Internet Engineering Task Force, May 1997.

[85] P. Sharma, D. Estrin, S. Floyd, V. Jacobson, P. Sharma, D. Estrin, S. Floyd, and V. Jacobson, “Scalable timers for soft state protocols,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Kobe, Japan), p. 222, Apr. 1997.

[86] S. Raman and S. McCanne, “A model, analysis, and protocol framework for soft state-based communication,” in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Cambridge, Massachusetts), August/September 1999.

[87] M. McKusick, B. K., M. Karels, and J. S. Quarterman, *The Design and Implementation of the 4.4BSD UNIX Operating System*. Massachusetts: Addison-Wesley Publishing Company, 1996.

[88] D. Katz, “IP router alert option,” RFC 2113, Internet Engineering Task Force, Feb. 1997.

[89] C. Partridge and A. Jackson, “IPv6 router alert option,” RFC 2711, Internet Engineering Task Force, Oct. 1999.

[90] P. Pan and H. Schulzrinne, “PF_IPOPTION: A kernel extension for IP option packet processing,” Technical Memorandum 10009669-02TM, Bell Labs, Lucent Technologies, Murray Hill, NJ, June 2000.

[91] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei, “An architecture for wide-area multicast routing,” in *SIGCOMM Symposium on Communications Architectures and Protocols*, (London, UK), pp. 126–135, Sept. 1994.

[92] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang, “Reliable multicast framework for light-weight sessions and application level framing,” in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Cambridge, Massachusetts), Sept. 1995.

[93] W. Fenner, “Internet group management protocol, version 2,” RFC 2236, Internet Engineering Task Force, Nov. 1997.

[94] L. Delgrossi, L. Berger, and Eds, “Internet stream protocol version 2 (ST2) protocol specification - version ST2+,” RFC 1819, Internet Engineering Task Force, Aug. 1995.

[95] T. Ballardie, P. Francis, and J. Crowcroft, “Core based trees (CBT),” in *SIGCOMM Symposium on Communications Architectures and Protocols* (D. P. Sidhu, ed.), (San Francisco, California), pp. 85–95, ACM, Sept. 1993. also in *Computer Communication Review* 23 (4), Oct. 1992.

[96] NLANR, “NLANR network traffic packet header traces.” http://moat.nlanr.net/Traces/.

[97] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, “RSVP-TE: extensions to RSVP for LSP tunnels,” RFC 3209, Internet Engineering Task Force, Dec. 2001.

[98] L. Gao and J. Rexford, “Stable internet routing without global coordination,” in *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, June 2000.

[99] T. Li, “MPLS and the evolving internet architecture,” *IEEE Communications Magazine*, vol. 37, Dec. 1999.

[100] D. O. Awduche, “MPLS and traffic engineering in IP networks,” *IEEE Communications Magazine*, vol. 37, Dec. 1999.

[101] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, “Deriving traffic demands for operational IP networks: Methodology and experience,” in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Stockholm, Sweden), August/September 2000.

[102] ITU-T, “Recommendation Q.2931, B-ISDN. digital subscriber signalling system no. 2 (DSS 2). user-network-interface (UNI) layer 3 specification for basic call/connection control,” tech. rep., ITU Telecommunication Standardization Sector, Apr. 1993.

[103] A. Banerjea, D. Ferrari, B. A. Mah, and M. Moran, “The tenet real-time protocol suite: Design, implementation, and experiences,” Technical Report TR-94-059, University of California at Berkeley, Berkeley, California, Nov. 1994.

[104] S. McCanne, V. Jacobson, and M. Vetterli, “Receiver-driven layered multicast,” in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Palo Alto, California), pp. 117–130, Aug. 1996.

[105] P. M. Merlin and P. J. Schweitzer, “Deadlock avoidance in store-and-forward networks - I: store-and-forward deadlock,” *IEEE Transactions on Communications*, vol. COM-28, pp. 345–354, 1980.

[106] P. M. Merlin and P. J. Schweitzer, “Deadlock avoidance in store-and-forward networks - other deadlock types,” *IEEE Transactions on Communications*, vol. COM-28, pp. 355–360, 1980.

[107] I. S. Gopal, “Prevention of store-and-forward deadlock in computer networks,” *IEEE Transactions on Communications*, vol. C-33, no. 12, pp. 1258–1264, 1985.

[108] I. Cidon, J. M. Jaffe, and M. Sidi, “Distributed store-and-forward deadlock detection and resolution algorithms,” *IEEE Transactions on Communications*, vol. COM-35, no. 11, pp. 1139–1145, 1987.

[109] International Telecommunication Union, “Visual telephone systems and equipment for local area networks which provide a non-guaranteed quality of service,” Recommendation H.323, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, May 1996.

[110] S. McCanne and V. Jacobson, “vic: A flexible framework for packet video,” in *Proc. of ACM Multimedia ’95*, Nov. 1995.

[111] V. Jacobson, “Multimedia conferencing on the Internet,” in *SIGCOMM Symposium on Communications Architectures and Protocols*, (London, England), Aug. 1994. Tutorial slides.

[112] I. Kouvelas, V. Hardman, and A. Watson, “Lip synchronisation for use over the Internet: Analysis and implementation,” in *Proceedings of the IEEE Conference on Global Communications (GLOBECOM)*, (London, England), Nov. 1996.

[113] H. Schulzrinne, “Voice communication across the Internet: A network voice terminal,” Technical Report TR 92-50, Dept. of Computer Science, University of Massachusetts, Amherst, Massachusetts, July 1992.

[114] S. Jamin, *A measurement-based admission control algorithm for integrated services packet networks*. PhD thesis, Dept. of Computer Science, University of Southern California, Los Angeles, California, Aug. 1996.

[115] L. Breslau, E. Knightly, S. Shenker, I. Stoica, and H. Zhang, “Endpoint admission control: Architectural issues and performance,” in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Stockholm, Sweden), Aug. 2000.

[116] V. Elek, G. Karlsson, and R. Ronngren, “Admission control based on end-to-end measurements,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Tel Aviv, Israel), Mar. 2000.

[117] D. L. Mills, “Network time protocol (version 3) specification, implementation,” RFC 1305, Internet Engineering Task Force, Mar. 1992.

[118] H. Schulzrinne, “RTP profile for audio and video conferences with minimal control,” RFC 1890, Internet Engineering Task Force, Jan. 1996.

[119] K. Nichols, S. Blake, F. Baker, and D. Black, “Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers,” RFC 2474, Internet Engineering Task Force, Dec. 1998.

[120] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, “Assured forwarding PHB group,” RFC 2597, Internet Engineering Task Force, June 1999.

[121] V. Jacobson, K. Nichols, and K. Poduri, “An expedited forwarding PHB,” RFC 2598, Internet Engineering Task Force, June 1999.

[122] S. Kent and R. Atkinson, “Security architecture for the internet protocol,” RFC 2401, Internet Engineering Task Force, Nov. 1998.

[123] W. R. Stevens, *TCP/IP illustrated: the implementation*, vol. 2. Reading, Massachusetts: Addison-Wesley, 1994.

[124] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. New York: McGraw-Hill, 1990.

[125] H. Schulzrinne and Columbia University IRT Lab, "rtptools: Tools for RTP." http://www.cs.columbia.edu/IRT/software/rtptools/.

[126] W. Almesberger, J.-Y. L. Boudec, and T. Ferrari, "Scalable resource reservation for the Internet," in *Proc. of IEEE Conference Protocols for Multimedia Systems – Multimedia Networking (PROMS-MmNet)*, (Santiago, Chile), Nov. 1997. Technical Report 97/234, DI-EPFL, Lausanne, Switzerland.

[127] S. Jamin, S. J. Shenker, and P. B. Danzig, "Comparison of measurement-based admission control algorithms for controlled-load service," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Kobe, Japan), Apr. 1997.

[128] K. Ramakrishnan and S. Floyd, "A proposal to add explicit congestion notification (ECN) to IP," RFC 2481, Internet Engineering Task Force, Jan. 1999.

[129] G. Feher, K. Nemeth, M. Maliosz, I. Cselenyi, J. Bergkvist, D. Ahlard, and T. Engborg, "Boomerang - a simple protocol for resource reservation in ip networks," in *IEEE Workshop on QoS Support for Real-Time Internet Applications*, (Vancouver, Canada), June 1999.

[130] Sprint, "Sprintlink policies." http://www.sprintlink.net/policies.htm.

[131] C. Labovitz, G. Malan, and F. Jahanian, "Origins of internet routing instability," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (New York), Mar. 1999.

[132] O. Schelen and S. Pink, "Aggregating resource reservation over multiple routing domains," in *Proc. of Fifth IFIP International Workshop on Quality of Service (IwQOS)*, (Cambridge, England), June 1998.

[133] S. Berson and S. Vincent, “Aggregation of internet integrated services state,” in *IWQOS*, 1998.

[134] A. Terzis, L. Wang, J. Ogawa, and L. Zhang, “A two-tier resource management model for the internet,” in *Proceedings of Global Internet*, Dec. 1999.

[135] R. Guerin, S. Herzog, and S. Blake, “Aggregating RSVP-based QoS requests,” Technical Report, University of Pennsylvania, Nov. 1997.

[136] F. Baker, C. Iturralde, F. L. Faucheur, and B. Davie, “Aggregation of RSVP for IPv4 and IPv6 reservations,” RFC 3175, Internet Engineering Task Force, Sept. 2001.