

MP 00W0000097

MITRE PRODUCT

---

# **A New Approach for Providing Quality of Service (QoS) in a Dynamic Network Environment**

**June 2000**

Duncan Thomson

Nancy Schult

Mohammad Mirhakkak

**Sponsor:** MITRE  
**Dept. No.:** W15F

**Project No.:** 51MSR86X-AA

Approved for public release; distribution unlimited.

©2000 The MITRE Corporation

**MITRE**  
Washington C3 Center  
McLean, Virginia

## Abstract

*This paper examines issues involved in providing Quality-of-Service (QoS) support in a dynamic network environment. Sources of network dynamics, in particular wireless links and mobile nodes, are discussed. The impact of these dynamics on network layer QoS mechanisms are discussed to explain the difficulties involved in attempting to apply these mechanisms in such an environment. A new paradigm for providing resource reservation-based QoS support is presented as a solution to some of the problems of providing QoS in a dynamic environment. In this paradigm, resource reservations represent ranges, and applications adapt to an allocated level of QoS provided by the network at some point within the requested range. This paradigm is explored in some detail, and is compared to related work by other authors. Based on the new paradigm we define a new protocol called dynamic Resource Reservation Setup Protocol (dRSVP). Finally, we describe the experience we have gained through an implementation of this protocol in a dynamic networking testbed environment, complete with adaptive streaming video and audio applications.*

KEYWORDS: Nomadic Network, Quality of Service, QoS, RSVP, Wireless, Dynamic Network, Prototype



# Table of Contents

Section	Page
<b>1. Introduction</b>	<b>1-1</b>
<b>2. Dynamic Network Characteristics</b>	<b>2-1</b>
2.1 Variable Link Characteristics	2-1
2.2 Node Movement	2-3
2.3 Variable Application Demand	2-4
<b>3. The Dynamic QoS Approach</b>	<b>3-1</b>
3.1 Reservations as Service Volumes	3-1
3.2 Reservations as Bandwidth Ranges	3-2
3.3 Implications for Applications	3-3
<b>4. Related Work on QoS in a Dynamic Network Environment</b>	<b>4-1</b>
<b>5. The Dynamic RSVP (dRSVP) Protocol</b>	<b>5-1</b>
5.1 dRSVP Protocol Operation Overview	5-1
5.2 dRSVP Messages	5-3
5.2.1 Ranges in Path Messages	5-4
5.2.2 Ranges in Resv Message	5-5
5.2.3 Measurement Specification in Resv Message	5-5
5.2.4 ResvTear Message	5-7
5.2.5 ResvNotify Message	5-7
5.2.6 Other Messages	5-8
5.3 Dynamic RSVP Message Processing	5-8
5.3.1 Path processing	5-8
5.3.2 Processing Reservation Messages	5-10
5.3.3 Processing for ResvNotify Messages	5-12
5.4 Admission Control	5-12
5.5 Bandwidth Allocation Algorithm	5-14
5.5.1 Flowspec Aggregation	5-14
5.5.2 External Allocation	5-14
5.5.3 Bandwidth Allocation Algorithm	5-15
5.5.4 Analysis of the Bandwidth Allocation Algorithm	5-17
5.6 dRSVP Application Programming Interface	5-18
<b>6. Test and Evaluation</b>	<b>6-1</b>
6.1 Test Environment	6-1

6.2 Test Applications	6-1
6.3 Testbed	6-5
<b>7. Conclusions</b>	<b>7-1</b>
<b>List of References</b>	<b>RE-1</b>
<b>Appendix A. Implementation Notes</b>	<b>A-1</b>

## Section 1

# Introduction

This paper presents an approach for providing Quality-of-Service (QoS) in a dynamic network environment. The paper also describes an implementation of this approach, which we developed by extending the Resource Reservation Setup Protocol (RSVP). In addition, the paper describes adaptive streaming video and audio applications that we have used to demonstrate and evaluate the benefits of our approach.

Our approach, which we refer to as “Dynamic QoS”, is a resource reservationbased approach to QoS that fits within the Integrated Services Internet (intserv) architecture[30]. However, we use an expanded notion of the meaning of the term “reservation”. With Dynamic QoS, a resource reservation request specifies a range of values, and the network makes a commitment to provide service at a specified point within this range. Applications request QoS by specifying the minimum level of service they are willing to accept and the maximum level of service they are able to utilize, and then adapt to the allocation within this range provided by the network, which may change with time.

Treating reservations as ranges and providing a mechanism for the network to signal the current allocation within the range, together provide the flexibility needed for operation in a dynamic environment in which network conditions change. As several authors have pointed out (e.g. [1]), the rapid deployment of wireless and mobile networking technology creates numerous examples of such dynamic networking environments. Section 2 of this paper discusses in more detail the dynamic characteristics at the physical, link, and network layers, that can be expected in networks with wireless links and mobile nodes. The difficulties of applying a resource reservation-based approach to QoS in such environments are discussed.

In Section 3 we describe our approach to tackling the problem of providing QoS support in these environments, based on the concept of treating reservations as ranges. We discuss this concept in some generality, and describe the semantics that we apply to the concept of a reservation range. We also discuss the implications on applications, which must be capable of expressing reservation requests as a range of desired values and of adapting to changes in the allocation provided by the network.

The notion of treating resource reservations as ranges and adaptively adjusting QoS within this range, was introduced in by Lu and Bharghavan [2], [3]. Other projects, for example the Mobiwire effort [4], have also explored the notion of QoS ranges, adaptivity, and other mechanisms for providing QoS in a mobile or wireless environment. More recently, the INSIGNIA protocol [5], combines the notion of QoS ranges with lightweight signaling carried in the data packet headers as an approach to providing QoS in a mobile ad hoc network. In Section 4 we survey and briefly examine these and other efforts, and highlight the differences between those efforts and our Dynamic QoS approach. The unique features

of our work include support for multicast flows, a new bandwidth allocation algorithm, and a new protocol design.

In Section 5 we provide the details of our implementation of the Dynamic QoS approach in a new network protocol that we call Dynamic RSVP (dRSVP), which is an extension of the resource reservation setup protocol (RSVP) [6], [7]. Extensions to the RSVP messages, new processing rules, the dRSVP bandwidth allocation algorithm, and a new application programming interface (API) are presented.

In Section 6 we describe our testbed and the adaptive streaming video and audio applications that work with the new API. We also describe our test network, which includes software emulation of variable speed links. Efficiency and scalability issues are touched upon, although detailed quantitative analysis of this area remains a topic for future work.

Finally, in Section 7, we conclude by summarizing what we have accomplished and demonstrated to date and by raising issues for further study.

## Section 2

# Dynamic Network Characteristics

The objective of our research has been to address the problem of providing QoS to support “nomadic computing,” which is defined by Bagrodia, Chu, Kleinrock, and Popek [8] as:

*"The support needed to provide a rich set of computing and communication capabilities and services to the nomad as he, she, or it moves from place to place, in a transparent, integrated, and convenient form."*

A key characteristic of networks designed to support nomadic computing is that the presence of mobile and wireless links creates a dynamic network environment. In this section, we define more carefully what we mean by “dynamic network environment.” We examine the sources of dynamics, the effects on different layers within the network, and the problems that this type of environment creates for conventional approaches to QoS.

Networks designed to support nomadic computing exhibit the following dynamics:

- 1) Dynamics caused by variable link characteristics, especially due to wireless links,
- 2) Dynamics caused by node movement, and
- 3) Dynamics caused by variable application demand.

These are each examined below.

## 2.1 Variable Link Characteristics

While links between network nodes are often treated as if they have fixed characteristics (e.g., bandwidth, error rate), in fact most links have characteristics that change with time. This is especially true for wireless links, which are subject to variations in transmission quality due to factors such as interference and fading. (In military networks other factors such as jamming must also be considered.) We should note that variable link characteristics also occur in wired links. In fact, V.90 modems, rapidly becoming the technology of choice for end-user internet access, are a prime example of variable link characteristics: V.90 modems will not only negotiate the data rate at connect time, but may also renegotiate the data rate during the course of the connection [9].

To understand how variable link characteristics at the physical layer will affect a QoS mechanism, we must examine how this variability will affect the link layer and the network layer. Assuming that changes in transmission quality are not handled by the physical layer (for example by increasing transmit power), the result observed by the link layer will be changes in bit error rate. The effect observed by the network layer depends on whether or not the link layer mechanisms in use recognize and respond to variations in error rate.

First, let us consider the case where the link layer does not detect or respond to the change in bit error rate. In this case, when the link degrades the network layer sees an increase in lost or corrupted packets, as packets are never received at all, are dropped by the link layer with frame checksum errors, or are received by the link layer and passed up to the network layer with undetected and uncorrected bit errors. When this happens, depending on the network layer protocols in use, corrupted packets may be dropped at the network layer with checksum errors, or they may be forwarded on through the network and passed to the transport layer, where the errors may or may not be detected and dealt with depending on the transport layer protocol or application layer protocol in use.

In this case the network layer will experience higher rates of packet loss and corruption, but the interface bandwidth perceived by the network layer will not change. This situation would significantly complicate the design of a network layer QoS solution. Some form of network layer error detection and correction would probably be required to provide QoS assurances, and it would be difficult for the network layer to distinguish between packet loss due to congestion and loss due to link layer corruption<sup>1</sup>. It would also be difficult for the network layer to determine the current available bandwidth, which is a key parameter in any resource reservation-based QoS mechanism. For these reasons, we believe that it is preferable to deal with variable bit error rate within the link layer.

Therefore, let us assume that the link layer reacts to the change in link error rate. Several different types of reaction are possible. For example, if the link layer protocol includes automatic repeat-request (ARQ), then, as transmission quality decreases, the number of retransmissions will increase, and the main impact on the network layer will be a decrease in the effective throughput of the link. A sophisticated link layer could also employ an adaptive error correction mechanism to increase or decrease the amount of error correction coding in response to changes in transmission quality. Products that do this are not widespread, however they do exist. For example, Scientific Research Corp. has produced a high-speed wireless modem that adaptively applies forward error correction (FEC), depending on link conditions [10]. Adaptive FEC has also been demonstrated in a mobile wireless environment in the Mobeware project, described in [4]. The impact on the network layer will be variation in effective delay or throughput, depending on the coding algorithm. The link layer could also react by changing its modulation technique. Advanced 802.11 products, for example

---

1 Lacking link layer mechanisms, it would be possible to include mechanisms in the network layer protocols that detect and respond to variations in error rates due to changes in transmission quality, and we did at one point give some consideration to implementing such algorithms in our testbed. However, it became clear to us that this approach was complex and impractical, due to the difficulty of distinguishing within the network layer between loss due to network congestion and loss due to transmission quality problems. Furthermore, response to transmission quality problems seems more properly to belong in the link layer, and, as we have seen, is in fact present in the link layers for most media where these problems can be expected.

Lucent Technologies' "WaveLAN Turbo" wireless network interface cards, are capable of operating with different modulation techniques, at speeds ranging from 1 to 11 Megabits per second (Mbps), depending on observed transmission characteristics.

From the above, we observe that as the link layer reacts to variable bit error rate, the main impact observed by the network layer is a change in effective throughput, while packet loss due to corruption remains low. This is significant, because interface bandwidth is a key parameter in any resource reservation-based QoS approach. Current link throughput information must be made available to the network layer software whenever conditions change, preferably by explicit signaling from the link layer. In current implementations interface speeds are likely to be treated as a configuration parameter, which is read when the system is booted or a daemon is started and never checked again. Clearly, this is inadequate for the environment being considered. In order to appropriately perform admission control, allocate resources, and other functions necessary for providing QoS, the network layer needs updated information from the link layer on the effective data rate of each interface, as well as possibly other parameters such as latency. As part of the DARPA funded Global Mobile (GloMo) effort, a general framework for internet devices operating over wireless links was developed [11]. This framework defines interfaces between different layers or components and includes the ability of the network layer to obtain information on current conditions observed by the link layer, including current link speed.

Another possible architecture would be for a subnet bandwidth manager such as that described in [12] or [13] to assume responsibility for controlling link layer QoS and determining available bandwidth on an interface. The subnet bandwidth manager would interact with a network layer QoS mechanism and provide information on available bandwidth. The subnet bandwidth manager also deals with link layer QoS and resource management in a shared media environment, which is not addressed further in this paper.

## **2.2 Node Movement**

An obvious source of network dynamics in nomadic networks is node movement, which has several consequences. First, it exacerbates the problem of variable link characteristics discussed above, as nodes move in and out of areas of good signal strength. Another consequence of node movement is that nodes may have to switch to different media as they move in and out of coverage. For example, an aircraft sitting at the gate could be connected to a high speed wireless Local Area Network (LAN) link, switch to Very High Frequency (VHF) data radio connectivity after pulling back from the gate, and finally switch to satellite connectivity when out of radio range over the ocean. Alternatively, a laptop or Personal Digital Assistant (PDA) could be connected to a wireless LAN within a building, then switch over to Personal Communication System (PCS) or cellular connectivity when moving outside. A "vertical handoff" approach has been described by Stemm and Katz [1], in which seamless connectivity to mobile nodes is maintained by handing off between small cells with high bandwidth and wide area cells with lower bandwidth.

Node movement also means that the network topology can change. In the simple case, this consists of the movement of end systems through a fixed network infrastructure. Mobile end systems are “handed off” between fixed access points. In the more general case of a mobile ad hoc network, intermediate systems (routers) also move, possibly causing relatively rapid routing changes. Clearly, this has a major impact on a resource reservation-based QoS approach, as resources that were available and reserved to support a reservation along one route may not be available on the new route. Various approaches (discussed in Section 3) have been proposed for using “pre-reservation” (in the simpler “handoff” case), or “standby routes” (in the more complex mobile ad-hoc network case) to allow the network to make a QoS commitment that can be honored even when the network topology changes. However, it seems intuitively clear that to fully ensure that a fixed level QoS can be maintained in a mobile environment would result in prohibitive under-utilization of the network. This is because, to allow totally general mobility with a guarantee of no loss of QoS, it would be necessary to reserve resources on *every* link for *every* flow. Therefore, we believe that a “weaker” approach, such as our “Dynamic QoS” approach, is more appropriate in a mobile environment. Our approach does not attempt to guarantee that a fixed level QoS will be preserved through topology changes. Instead, when topology changes occur, our approach allows the new level of available resources to be discovered and for QoS levels to be adjusted accordingly.

### **2.3 Variable Application Demand**

Another source of dynamics is one that is common to both nomadic and fixed environments. That is: variable demand on network resources by end-system applications. The conventional response to variable application demand is based on admission control. That is, some end users may be denied access in order to preserve QoS for those that have been admitted. This results in some users being granted the requested QoS, and others being denied service or forced to use a lower grade service-model, for example best-effort. Admission control may be performed on a “first come, first served” basis, or it may include priority and preemption mechanisms to implement some desired policy.

Admission control is a valuable tool for dealing with variable application demand for QoS, because it is generally better to deny service to some users in order to grant service to others, rather than providing an unacceptable QoS to all users. The limitation with this approach, however, is that it traditionally provides an “all or nothing” service. For some applications, it may be desirable to provide a reduced level of QoS to a larger number of users. In order to do this, the network and applications need some way to communicate on what levels of QoS would be acceptable and can currently be supported. In addition to providing a means for dealing with network dynamics due to wireless links and node movement as discussed above, we will show how the “Dynamic QoS” approach also has the significant benefit of allowing more flexible sharing of available QoS resources among applications.

## Section 3

# The Dynamic QoS Approach

In this section we provide a general discussion of the approach we have taken to tackle the problems described above. Details of the protocol that implements this approach are provided in Section 5.

### 3.1 Reservations as Service Volumes

In conventional QoS mechanisms, such as internet integrated services (intserv) or Asynchronous Transfer Mode (ATM), a reservation can be represented by point in an  $n$ -dimensional space, with coordinates defining the characteristics of the service. For example, with the Controlled-Load service in an intserv network a reservation contains an  $n$ -tuple defining the token bucket traffic specification parameters  $(r, b, p, m, M)$ , where  $r$  is the average rate,  $b$  is the token bucket depth,  $p$  is the peak rate,  $m$  is the minimum policed unit, and  $M$  is the maximum packet size [14]. Other network types and QoS service models may use other parameters, such as maximum delay, delay jitter, packet loss rate, and so on, to specify a reservation. Acceptance of a reservation means that the network makes a “commitment” to provide the service with characteristics specified by the reservation  $n$ -tuple<sup>2</sup>.

The problem with representing a reservation as a single point is that a binary decision must be made: either service can be provided as requested, or no service commitment (beyond best-effort) is given at all. This does not allow the flexibility needed for operating within a dynamic network environment; it does not allow any way to respond to varying network resources, and it does not allow any way to respond to varying application demand on available resources.

To solve this problem, the concept of a reservation can be expanded. In our approach, a reservation is represented, in the most general terms, by a volume of  $n$ -dimensional space representing the set of possible service levels that are utilizable by the end system. Acceptance of a reservation means that the network commits to provide service at some point, which will be specified by the network, within the requested volume. We use the term

---

<sup>2</sup> In the Controlled Load service model, acceptance of a reservation with these parameters means that the network can provide service equivalent to that which could be expected on an uncongested network, as long as the traffic flow does not exceed the token bucket specification. In the Guaranteed service model the acceptance of a reservation with these parameters, with a specified reservation rate  $R$  and slack term  $S$ , means that the network can provide end-to-end behaviour conforming to the fluid model, with a known (computable) maximum delay, as long as the offered traffic does not exceed the token bucket specification.

“*reservation range*” to refer to the volume in n-space representing the service requested by the application, and the term “*allocation*” to refer to the point within this volume that corresponds to the service that the network commits to provide.

By treating a reservation as a request for service somewhere within a specified volume, we create the flexibility needed to deal with all of the types of network dynamics discussed in the previous section. As available resources change, the network can readjust allocations within the reservation range. If resources decrease below the level currently allocated, the network can offer a more reasonable response than simply dropping the reservation, as long as there is some point within the reservation range that can be supported. Similarly, as the number of application flows competing for resources increases, rather than simply refusing to admit new flows or preempting existing flows, the network can attempt to adjust the allocation for all flows, so that each flow can be accommodated at some point within its requested range.

### 3.2 Reservations as Bandwidth Ranges

Although in general network service is specified by an n-tuple, for the Controlled-Load service model it is possible to focus on a single parameter – average data rate  $r$  – and to assume that the other parameters remain fixed. This allows us to greatly simplify our study and, even more, to simplify our initial implementation. For our implementation, which uses Class-Based Queuing (CBQ) [15] and the Controlled Load service model, this simplification works well, since average data rate is in fact the critical managed resource. The CBQ implementation we have utilized in our testbed ([16], [17]) creates a separate queue for each flow, and services each queue in such a way as to attempt to provide that queue with its allocated share of the outgoing interface bandwidth. Little is done with the other parameters, although they are carried throughout the network and taken into account when reservations are aggregated. Furthermore, as discussed below, for the Real Time Protocol (RTP)-based streaming video and audio applications we have experimented with in our testbed, this turns out to be a very useful service model, and average transmission bandwidth turns out to be the critical reservation parameter.<sup>3</sup>

For other service models, further study is required to determine the extent to which this simplification can be applied. In particular, in a service model which guarantees zero packet loss, output buffer size becomes a critical managed resource, and therefore token bucket depth becomes a critical parameter in reservation requests. The network may find that it can meet its commitment at a given average transmission rate, but only if it is allowed to adjust the token bucket depth. In a service model that includes bounds on delay variation, peak transmission rate may become a critical parameter. In order to prevent jitter, a network node

---

<sup>3</sup> We are not unique in our focus on bandwidth as the key parameter to be managed in providing QoS. A similar focus can be found in nearly all the related works cited in this paper.

may need to respond to input bursts, up to the specified peak rate, by creating a corresponding output burst. Peak transmission rate would therefore become a critical resource to be managed on outbound interfaces, in addition to average transmission rate. In situations such as these, it may still be possible to simplify the model by assuming that only one reservation parameter (probably still average transmission rate) should be specified as a range, while the other values are specified as scalars. The network would have flexibility to adjust the allocation of one parameter within the reserved range, in order to maintain its commitments for all parameters. For example, jitter could be prevented by limiting average flow transmission rates to ensure sufficient spare transmit bandwidth to allow transmission of a burst on one flow without affecting other flows. Application of the concept of reservation ranges to other service models remains a topic for further study.

Throughout the remainder of this paper, we make the assumption that a reservation can be considered to be a bandwidth range (minimum to maximum usable bandwidth) and that other reservation parameters are specified as scalars.

### **3.3 Implications for Applications**

Obviously, treating reservations as ranges has implications for applications. First, the application must have some notion of the QoS range within which it can operate. At first glance, this may seem to add complexity, but in fact we believe that in most cases this actually simplifies the task of the application programmer. When a QoS request must specify a scalar value, the application programmer has no way to know what this value should be. The solutions to this problem are not very good. For example, the application could be programmed to start out requesting the maximum level of service that the application might want to use, and if this request fails admission control, continue to make requests for successively lower levels of service until one request succeeds. Another approach might be to have the application prompt the user for the speed at which the end system is connected to the internet (e.g., dial-up modem, Integrated Services Digital Network (ISDN) line, Digital Subscriber Line (DSL), cable modem)<sup>4</sup> and then, assuming that this will be the bottleneck, base the requested reservation request on this value. Clearly, both of these approaches have disadvantages. On the other hand, if reservations are treated as ranges, the application programmer has a reasonable chance of being able to determine a priori the minimum service level at which the application can reasonably operate and the maximum service level that it could utilize. These values can then be programmed in (or configured by the user according to his intended use of the application) as the QoS range that application will use in reservation requests. Then, at run time, when a QoS request is made, the network

---

<sup>4</sup> This is a fairly common practice, for example, web-based servers for downloading application packages servers often prompt the user for the connection speed, and then customize the package to be downloaded to provide the maximum functionality within an “acceptable” download time.

will provide feedback on the current allocation within this range that can be supported, and the application can react accordingly.

For example, consider a web server that users hit to obtain a software package or “plug in”, where the server has several different versions of the package, and the server administrators have some notion of the download times that users will consider acceptable. If the smallest version of the package is 100 Kilobytes (KB) and 1 minute is the maximum time we want any user to have to wait for the download, then we know that the minimum bandwidth needed is  $800,000\text{bits}/60\text{ seconds}$ , or 13 Kbps. If the largest version of the package is 1 MB, and we know that 1 second is the download time beyond which improvements will not even be noticed by the user, then we know the maximum bandwidth needed is 8 Mbps. The QoS range then becomes (13 Kbps, 8 Mbps), and the version of the package that is downloaded to the user can be selected by the server at connect time based on the current available bandwidth within this range. Many other examples can be constructed to make the point that reservations as bandwidth ranges is, in fact, much more suitable from the point of view of the application programmer than having to pick a single value.

The second implication is that applications should be capable of adapting their behavior based on the current allocation, which must be provided as feedback from the network to the application. This does require additional sophistication on the part of the application, but is not, we believe, an unreasonable task. The example discussed above indicates how this might be done in a web server, which simply selects appropriate content to download based on the available allocation. We are currently collaborating with another group to implement exactly this type of adaptive web server. Later in the paper we will discuss how we added adaptive behavior to streaming audio and video applications, which dynamically adjust the encoding scheme (for audio) and frame rate (for video) in response to changing allocations within the reservation range.

Finally, in order to support the inclusion of range-based reservations, and adaptation to changing network allocation, the application program interface (API) must be extended to support this new paradigm.

## Section 4

# Related Work on QoS in a Dynamic Network Environment

A survey of the literature reveals that there are three problems that are often brought up when attempting to provide QoS in a wireless and/or mobile network environment:

- 1) *The QoS routing problem*: How to find a route through the network that is capable of supporting a requested level of QoS.
- 2) *The QoS maintenance problem*: How to ensure that, when network topology changes, new routes that can support existing QoS commitments are available, or can be quickly found.
- 3) *The variable resource problem*: How to respond to changes in available resources, either as the result of a route change, or as the result of changes in link characteristics within a given route.

These issues are obviously very closely related, but we have found it useful to distinguish among them when examining the work in this area. In particular, we believe that the QoS routing problem should be de-coupled from the other problems, and that a solution to the variable resource problem can obviate the need to solve the QoS maintenance problem. We will elaborate on these points as we examine related work in this area by other researchers.

Work on the QoS routing problem in mobile ad hoc networks has been documented by Chen and Nahrstedt [18]; Lin and Liu [19]; and Sivakumar, Shiha, and Bharghavan [20]; among others. We do not discuss these papers here, since our work has not centered on this problem.

Solving the QoS maintenance problem in a general mobile ad hoc network is extremely difficult, and the published body of work in this area has generally assumed networks in which only the end systems move. The QoS maintenance problem is then reduced to a QoS handoff problem; that is: how to maintain QoS when an end system node is handed off from one access node to another. The QoS handoff problem, though still challenging, is more tractable and has been tackled by several researchers. One significant effort tackling the QoS maintenance problem (for mobile end systems) is that of Talukdar, Badrinath, and Acharya, at Rutgers University [21]. The solution proposed by these authors assumes that, at least for some mobile users, mobility will be predictable. That is, the mobile end systems will be able to provide a mobility specification that defines which access points they will visit. Three new mobility-related service models are proposed: Mobility Independent Guaranteed (MIG), Mobility Independent Predictive (MIP), and Mobility Dependent Predictive (MDP) service. With MIG and MIP service, mobile end systems are provided guaranteed or predictive

service, respectively, as long as they move in accordance with their mobility specification. With MDP service, mobile end systems are provided a “weak” form of predictive service as long as they move in accordance with their mobility specification<sup>5</sup>. The MIG and MIP service are provided by preallocating resources for all routes needed by a mobile host, according to its predicted motion defined in its mobility specification. In order to avoid dramatic underutilization, the MDP service is provided using resources that have been allocated to other flows, but which are currently being underutilized. This means that a mobile node receiving MDP service may experience QoS degradation when the node moves into another cell, or when other nodes receiving MIG or MIP service move into the same cell. An implementation of the MIG, MIP and MDP service models, based on modified versions of Mobile IP and RSVP, is described in [22].

We observe that the MDP service suffers from the variable resource problem, and no solution for this problem is proposed in [21] or [22]. That is, there is no provision for varying link bandwidths, and no flexibility to adjust QoS levels to deal with changes in resources. An interesting and potentially valuable approach might be to combine the concept of adaptive QoS ranges with the MDP service model, creating a Mobility Dependent Predictive service with Adaptive QoS.

In [2] and [3], Lu, Lee, and Bharghavan propose a system for solving both the QoS maintenance problem (for the handoff case only) and the variable resource problem. The QoS maintenance problem is tackled by performing advance resource reservation in cells that are neighbors of the cell currently occupied by a mobile end system. The variable resource problem is tackled by adaptively re-adjusting the quality of service within prenegotiated bounds. We believe that this concept (treating reservations as ranges and allowing the network the flexibility to adjust QoS within this range) is crucial to solving the resource change problem, and we have adopted this concept as the basis for our work. However, we have interpreted the concept differently. In [2], the concept of QoS ranges is interpreted to mean that a QoS “guarantee” is provided for the low end of the range, and best effort service is provided beyond that point:

*“...service is guaranteed in the sense that each connection is guaranteed its minimum QoS requirement; the service is best effort in the sense that the network provides best-effort service beyond the minimum QoS support.”*

In contrast, our interpretation of the notion of QoS ranges is that the network provides service at a signaled QoS allocation point within the range requested in QoS reservation request. Service is “guaranteed”<sup>6</sup> at the allocated point, however the network may change

---

<sup>5</sup> Here the terms guaranteed and predictive are as defined by Clark, Shenker, and Zhang [23].

<sup>6</sup> We use the term “guarantee” loosely here; the exact meaning of the term “guarantee” depends on the service model.

this allocation point at any time. The application should be capable of adapting its transmission characteristics to make use of the allocated QoS. Our implementation provides a Controlled Load service, which means that if the application adjusts its transmission to stay within the current allocation, it will not experience congestion. In this regard, our approach could be considered a form of congestion control with explicit rate indication, as described by Charny, Clark, and Jain [24]. However, our approach could also be extended to other service models, for example, Guaranteed QoS [25].

In addition to a somewhat different interpretation of the concept of reservation ranges, the protocol mechanism and algorithms we have used to realize this concept are different from those presented in [2] and [3]. One significant difference is that our system supports multicast flows, which creates the requirement for aggregating information at branch points. Another major difference comes in the distributed bandwidth allocation algorithm. The bandwidth allocation algorithm presented in [2] is an adaptation of the explicit rate indication congestion control algorithm first presented by Charny et. al. in [24]. Our algorithm was developed independently as part of our research. Whereas the Lu-Bharghavan algorithm attempts to give each flow the minimum bandwidth requested, plus an equal amount of any “excess” bandwidth, our algorithm attempts to give each flow the minimum bandwidth requested, plus an equal fraction of the requested bandwidth range. Both algorithms have the concept of flow “bottlenecks”, but use different protocol mechanisms for identifying the bottlenecks. Preliminary analysis and lab experience indicate that our algorithm converges quickly (in one round trip), but more work remains to rigorously analyze the convergence performance of our algorithm.

A significant difference between our work and that presented by Lu, and Bharghavan in [2] is that our approach does not include an explicit solution to the QoS maintenance problem. The approaches presented in [2] and in the other papers discussed below may offer promise for solving this problem in the limited case where only end systems move (the QoS handoff problem). However, due to our special interest in military applications, we have targeted our work for use in networks where intermediate system nodes may move, that is: mobile ad hoc networks. In these networks, routing is a difficult problem, QoS routing is even more difficult, and a practical solution to QoS routing which will ensure that QoS can be maintained as new routes are selected is perhaps intractable. Therefore, we believe that it is important to decouple these problems. We do not attempt to tackle the QoS routing problem or the QoS maintenance problem. Rather, our approach relies on “periodic control messages being sent (as in RSVP)” to reestablish QoS along the new route when a change occurs. Then, we rely on our concept of adaptive QoS to deal with the fact that a different level of resources may be available along the new route. Thus we avoid the QoS maintenance problem, and our solution to the resource change problem nevertheless allows relatively seamless provision of QoS. However, the reliance on periodic control messages means that, when a route changes, there will be a period during which the traffic receives only best effort service. State refresh messages must propagate end to end (round trip) before QoS will be

reestablished, probably at a new allocation level, along the new route. Nevertheless, we believe that there is a large class of applications that can tolerate transient periods of degraded service, yet benefit from the Dynamic QoS mechanisms we have proposed.

The INSIGNIA in-band signaling system for supporting QoS in mobile ad hoc networks [5] also relies on soft-state and adaptivity for dealing with the resource change problem. As with our Dynamic QoS approach, INSIGNIA includes mechanisms for signaling QoS requirements along the new route, and adapting to changes in available resources. By incorporating its signaling into the data packet headers, INSIGNIA has the significant advantage of potentially very rapid reestablishment of QoS along the new path. The concept of adaptability in INSIGNIA hinges on the data flow being hierarchical – consisting of a “base layer” and an “enhancement layer”. This may be suitable for certain important applications (in particular video), but not all application data lends itself to this kind of layering. We believe that the concept of bandwidth ranges, as proposed by Lu and Bharghavan and further developed in our work as well as in the Mobware effort (discussed below), offers considerably more generality. An interesting possibility would be to combine the in-band signaling of INSIGNIA with the concept of bandwidth ranges.

The Mobware effort [4] has created an open and active programmable mobile testbed for experimenting with mobile and wireless networking. In the Mobware testbed CORBA is used to signal handoffs of WaveLAN-equipped mobile end systems connected to an IP-over-ATM wired network. The Mobware work does not explicitly tackle the QoS maintenance problem; that is, the pre-allocation approaches discussed earlier are not applied, so when a handoff occurs there is no guarantee that QoS will be maintained. However, the Mobware work does tackle the resource change problem. If available resources change (either because of a handoff between cells or because of transmission quality changes within a cell), there is a mechanism for dealing with this change. Application adaptation to varying QoS in the experiments described in [4] was performed by passing the application flows, which consisted of hierarchically encoded video streams, through a filter deployed at the access point. The filter will drop or add layers containing more or less of the video flow in response to changing QoS. We should also note that the Mobware work appears to be extremely general; the mechanism described above is just one approach that could be implemented in the Mobware testbed.

The Mobware work also further explored the notion of a utility-fair bandwidth allocation algorithm, proposed by Bianchi, Campbell, and Liao [26]. In our work, as in that of Lu, Lee, and Bharghavan [3], an application defines QoS in terms of a bandwidth range. Allocating bandwidth within this range can be done in different ways. The bandwidth algorithm described in [3] allocates bandwidth by giving each flow the minimum plus an equal amount of the “excess” bandwidth. Our algorithm allocates bandwidth by giving each flow the minimum plus an equal percentage of its requested range of bandwidth. The approach proposed in [26] and utilized within Mobware allows each flow to specify how much “utility” it will receive from each unit of extra bandwidth. “Excess” bandwidth is then

allocated so that each flow receives an equal “utility”. This approach could have benefit, in particular for applications whose utility functions are step functions (a multi-resolution hierarchical flow, for example, can only make use of extra bandwidth if it is sufficient to add another layer to the flow). However, whereas the CORBA-based signaling used in Mobeware can support the complexity of including an arbitrary function definition as part of an application’s reservation request, it is unclear whether this level of complexity could be supported with a more lightweight signaling mechanism. Furthermore, we should point out that the Mobeware work appears to deal only with the case where the bottleneck location in the network is known to be at the last hop from the access point to the mobile end system. This will clearly not be the case in a mobile ad hoc network, and it may not be the case in other network architectures, for example, a fixed topology network that includes a number of variable-quality wireless links internal to the network.

## Section 5

# The Dynamic RSVP (dRSVP) Protocol

To demonstrate the feasibility of the Dynamic QoS approach discussed in Section 3, we defined a distributed network protocol that we call dynamic RSVP or dRSVP. As the name suggests, this protocol is an extension of the resource reservation setup protocol (RSVP) [6]. We implemented this protocol by modifying and extending the University of Southern California (USC) Information Sciences Institute's (ISI) implementation of RSVP [27]. In this section we describe the RSVP protocol extensions and our implementation. (The description presented here assumes that the reader is familiar with the basic structure and functionality of RSVP [6], [7].)

The dRSVP protocol was created by making the following extensions and modifications to standard RSVP:

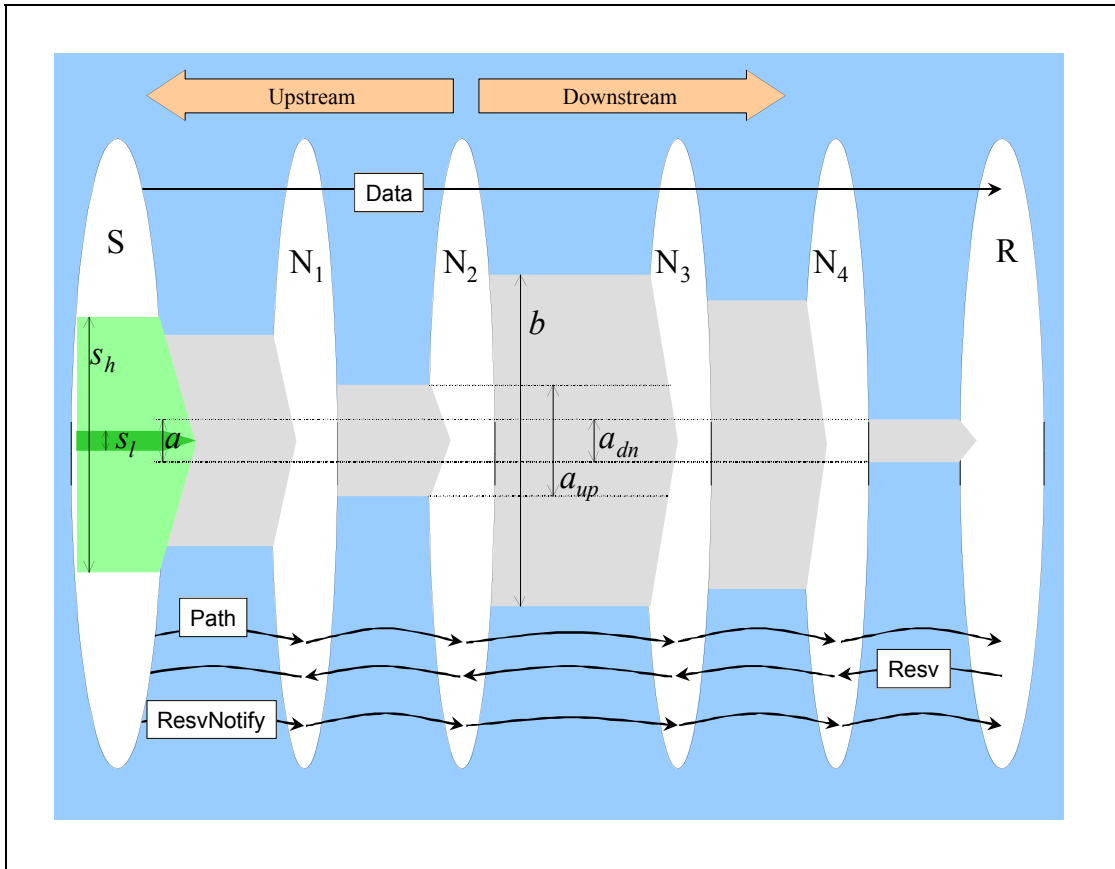
- a) Adding an additional flow specification (flowspec) in Resv messages and an additional traffic specification (tspec) in Path messages, so that they describe ranges of traffic flows,
- b) Adding a “measurement specification” (mspec) to the Resv messages, which is used to allow nodes to learn about “downstream” resource bottlenecks,
- c) Creating a new reservation notification (ResvNotify) message, which carries a “sender measurement specification” (smspec) information that is used to allow nodes to learn about “upstream” resource bottlenecks,
- d) Changing the admission control processing to deal with bandwidth ranges,
- e) Adding a bandwidth allocation algorithm that divides up available bandwidth among admitted flows, taking into account the desired range for each flow as well as any upstream or downstream bottlenecks for each flow,
- f) Extending the API to deal with bandwidth ranges.

These extensions and modifications to RSVP, which comprise the dRSVP protocol, are described below. First we provide an overview of the operation of the protocol. We then describe the dRSVP messages, dRSVP processing, and the new API.

## 5.1 dRSVP Protocol Operation Overview

Figure 5-1 illustrates a simple network in which node S sends data to node R through intermediate nodes  $N_1$ ,  $N_2$ ,  $N_3$ , and  $N_4$ . The nodes are connected by links, shown in the figure as wide bars, with the width of the bar corresponding to the bandwidth available on the link. The adaptive application running on node S can generate data at rates within the range from  $s_l$  to  $s_h$ . These values are communicated in Path messages, which flow through

the network hop by hop, following the same route as the data messages, to the receiver R. Upon receipt of the Path messages, the receiving application on R requests a reservation for this flow, with QoS range  $(s_l, s_h)$ . The request is carried through the network in Resv messages which reverse the route followed by the Path messages. Each node, upon receiving



**Figure 5-1. dRSVP Protocol Overview**

the Resv message, performs an admission control check and computes the bandwidth, within the range  $(s_l, s_h)$ , that it can allocate to the flow. Assuming the admission control test passes, the Resv message is propagated upstream towards S. The Resv message also contains a “receiver measurement specification” value, denoted  $m_r$ . The value of  $m_r$  is initialized at R to  $s_h$ , but as each node propagates the Resv message upstream, if the bandwidth allocation that it is able to give the flow is less than the received  $m_r$  value, it reduces the  $m_r$  value to the allocation. S applies similar logic to the Application Program Interface (API), treating the API as if it were an upstream link, so the application receives  $a$ , as shown, indicating the bandwidth that has been reserved end-to-end through the network. The application must

adapt to  $a$  in order to receive the agreed-upon service (controlled-load, in our implementation). If the reservation successfully propagates all the way through the network, node S initiates a ResvNotify message. This contains a “sender measurement specification”, denoted  $m_s$ , initialized at S to  $a$ . This propagates toward R and in a similar fashion each node limits the value of  $m_s$  to the bandwidth that it is able to allocate for the flow.

Since reservation requests are for ranges, rather than specific values, a key issue in dRSVP is what level of resources to allocate within the requested range. Let us consider how this decision is made at node  $N_2$ , which must determine how much bandwidth, out of a total of  $b$  on the link to  $N_3$ , to allocate to the flow. Within one round trip (after the reservation was issued at R)  $N_2$  has learned that there is a bottleneck somewhere downstream (on the link from  $N_4$  to R) with a value of  $a_{dn}$ , and that there is a bottleneck upstream (on the link from S to  $N_1$ ) with a value of  $a_{up}$ . Node  $N_2$  has total bandwidth  $b$  available on the link, so if this is the only flow present, it can allocate the maximum requested,  $s_h$ , but it need not reserve more than  $a_{dn}$ . If other flows are present, the dRSVP algorithm divides the available bandwidth among the flows, attempting to give each flow its minimum requested bandwidth, plus an equal fraction of its requested bandwidth range. This algorithm takes into account upstream and downstream bottlenecks for every flow, so that it avoids reserving resources for a flow that could not be utilized due to bottlenecks elsewhere in the network. The details of this bandwidth allocation algorithm are included in the following paragraphs.

Now consider what happens when the bandwidth on the link from  $N_4$  to R increases, as shown in Figure 5-2. The next Resv message propagated by  $N_4$  will allow all upstream nodes to learn that the bottleneck has been removed. The bottleneck is now the link between  $N_1$  and  $N_2$ . The figure shows the new values of  $a_{dn}$  and  $a_{up}$  that will be computed at node  $N_2$ .  $N_2$  now knows that this flow could now utilize up to  $a_{up}$ , as shown; remaining bandwidth can be freely allocated to other flows. Assuming that this is the only flow present, the bandwidth allocation provided to the application at S is now  $a_{up}$ , as shown.

## 5.2 dRSVP Messages

RSVP creates and maintains state information by periodically sending control messages in both downstream and upstream directions<sup>7</sup> along the data path for a session. These messages are encapsulated in IP packets and are captured and processed by each node along data paths. These messages are used to establish, modify or refresh states.

---

<sup>7</sup> Consistent with the RSVP terminology, the terms “sender,” “receiver,” “upstream,” and “downstream” are defined with respect to the *application* data flow; that is, senders are the nodes that transmit application data, which flows downstream through the network towards receivers. Note that some RSVP protocol messages flow “upstream” through the network from “receivers” towards “senders”.

The two primary RSVP messages that are used for these functions are Path and Resv. We have extended these messages, and added a new ResvNotify message.

### 5.2.1 Ranges in Path Messages

Path messages are initiated by data senders and travel downstream to receivers. A Path message can be addressed to a unicast or multicast address of a session and creates Path state

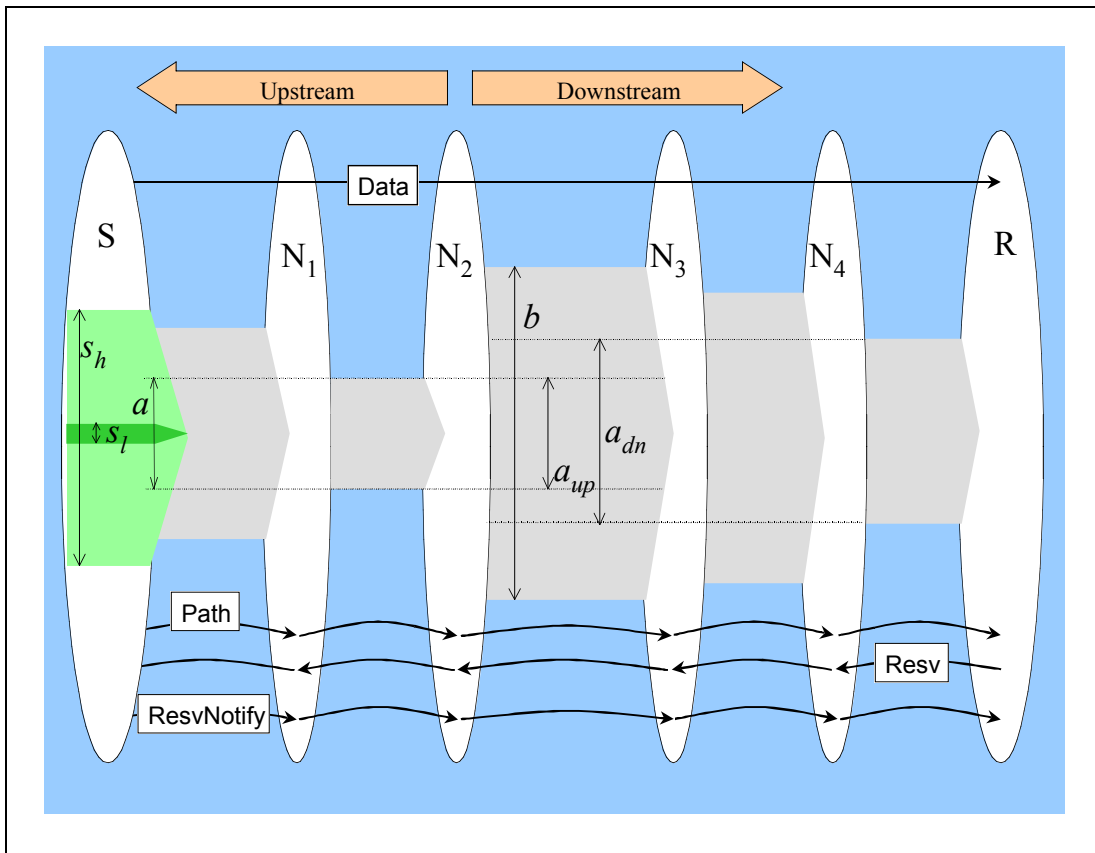


Figure 5-2. dRSVP Response to Link Bandwidth Changes

in each router between sender and receivers. The RSVP protocol processing module in each node (in our implementation rsvpd, the RSVP daemon) queries the routing table in order to forward a Path message along the route from a sender to receivers. In this way the Path messages flow through the network along the same route as the application data. Each Path message contains previous hop (PHOP) information, which is used to allow Resv messages (described below) to flow through the network in the reverse direction. The standard RSVP

Path message also contains a traffic specification (Tspec), which is used to characterize the traffic that the sender may generate.

In dRSVP, we wish to characterize the traffic as a range. To accomplish this, we need not one, but two Tspecs in the Path message. TspecLow defines the minimum flow that the sender may generate, and TspecHigh defines the maximum flow that the sender may generate.

The format of a dRSVP Path message is as follows:

```
<Path Message> ::= <Common Header> [ <INTEGRITY> ]
    <SESSION> <RSVP_HOP>
    <TIME_VALUES>
    [ <POLICY_DATA> ... ]
    [ <sender descriptor> ]
<sender descriptor> ::= <SENDER_TEMPLATE> <SENDER_TSPEC_LOW>
    <SENDER_TSPEC_HIGH>
    [ <ADSPEC> ]
```

Details of these fields can be found in [6]. The <SENDER\_TSPEC\_LOW> and <SENDER\_TSPEC\_HIGH> are of the type <SENDER\_TSPEC>, which is described in [6].

### 5.2.2 Ranges in Resv Message

Resv messages are initiated by receivers and are periodically sent upstream towards senders to perform bandwidth reservation. One of the fields in a Resv message is the flowspec that defines the requested level of QoS to be reserved. In dRSVP, we need two flowspecs in the Resv message, a high flowspec and low flowspec. These specify the upper bound and lower bound of the range of QoS requested for the flow, respectively.

When an application initiates a Resv message, it initializes these two fields. The application sets the high flowspec to the maximum data rate that it wishes to receive QoS support for. Similarly, the low flowspec is set to the minimum data rate. Normally, the high and low flowspec values would be set identical to the high and low Tspec values received in Path messages. The format of a dRSVP Resv message is found in the next section.

### 5.2.3 Measurement Specification in Resv Message

For dRSVP, we also added another field called the “measurement specification” (mspec) to each Resv message. Nodes use this field to learn the amount of resources that are available downstream.

The dResv message format is as follows:

```
<Resv Message> ::= <Common Header> [ <INTEGRITY> ]
```

```

    <SESSION> <RSVP_HOP>
    <TIME_VALUES>
    [ <Resv_CONFIRM> ] [ <SCOPE> ]
    [ <POLICY_DATA> ... ]
    <STYLE> <flow descriptor list>
<flow descriptor list> ::= <empty> |
    <flow descriptor list> <flow descriptor>

```

The following BNF rules specify in more detail the composition of a valid flow descriptor list for each of the reservation styles:

- Wildcard Filter (WF) Style:
 

```

<flow descriptor list> ::= <WF flow descriptor>
<WF flow descriptor> ::= <FLOWSPEC_LOW>
    <FLOWSPEC_HIGH>
    <MSPEC>

```
- Fixed Filter (FF) style:
 

```

<flow descriptor list> ::=
    <FLOWSPEC_LOW> <FLOWSPEC_HIGH>
    <MSPEC> <FILTER_SPEC> |
    <flow descriptor list> <FF flow descriptor>
<FF flow descriptor> ::=
    [<FLOWSPEC_LOW> <FLOWSPEC_HIGH>
    <MSPEC>] <FILTER_SPEC>

```
- Shared Explicit (SE) style:
 

```

<flow descriptor list> ::= <SE flow descriptor>
<SE flow descriptor> ::=
    [<FLOWSPEC_LOW> <FLOWSPEC_HIGH>
    <MSPEC>] <FILTER_SPEC> <filter spec list>
<filter spec list> ::= <FILTER_SPEC>
    | <filter spec list> <FILTER_SPEC>

```

These fields are described in [6]. <flow descriptor> is the one we modified for adaptive RSVP. The fields <FLOWSPEC\_LOW>, <FLOWSPEC\_HIGH>, and <MSPEC> are of the type <FLOWSPEC>, which is described in [6].

## 5.2.4 ResvTear Message

ResvTear deletes a specified reservation state. It travels upstream following the same Path as the corresponding Resv message. For dRSVP we did not change the format of the ResvTear message. However, additional code was added to create a ResvTear when network dynamics (decrease in link speed) causes an existing reservation to fail. In such a situation, a ResvTear is generated by the RSVP daemon to clear the corresponding reservation state.

## 5.2.5 ResvNotify Message

For dRSVP we added a new message called ResvNotify to the list of RSVP messages. The function of this message is to send information about available resources to downstream nodes. Sending this information about resources downstream causes the downstream nodes to avoid over-reservation for a flow that cannot use the reserved resources due to limitations in resources at upstream nodes.

The ResvNotify message format is as follows:

```
<ResvNotify Message> ::= <Common Header> [<INTEGRITY>]
    <SESSION> <RSVP_HOP>
    [ <SCOPE> ] <STYLE>
    <notify flow descriptor list>
<notify flow descriptor list> ::= <empty> |
    <notify flow descriptor list> <notify flow descriptor>
```

The following Backus-Naur Form (BNF) rules specify in more detail the composition of a valid <notify flow descriptor list> for each of the reservation styles:

- Wildcard Filter (WF) Style:  

```
<notify flow descriptor list> ::= <WF notify flow descriptor>
<WF notify flow descriptor> ::= <SENDER MSPEC>
```
- Fixed Filter (FF) Style:  

```
<notify flow descriptor list> ::=
    <SENDER MSPEC> <FILTER_SPEC> |
    <notify flow descriptor list> <FF notify flow descriptor>
<FF notify flow descriptor> ::=
    [<SENDER MSPEC>] <FILTER_SPEC>
```
- Shared Explicit (SE) style:  

```
<sender flow descriptor list> ::= <SE sender flow descriptor>
<SE sender flow descriptor> ::=
    [<SENDER MSPEC>] <FILTER_SPEC> <filter spec list>
```

```

<filter spec list> ::= <FILTER_SPEC>
                        | <filter spec list> <FILTER_SPEC>

```

As stated before, these fields are described in [6]. The field <SENDER MSPEC> is of the type <FLOWSPEC>, also described in [6].

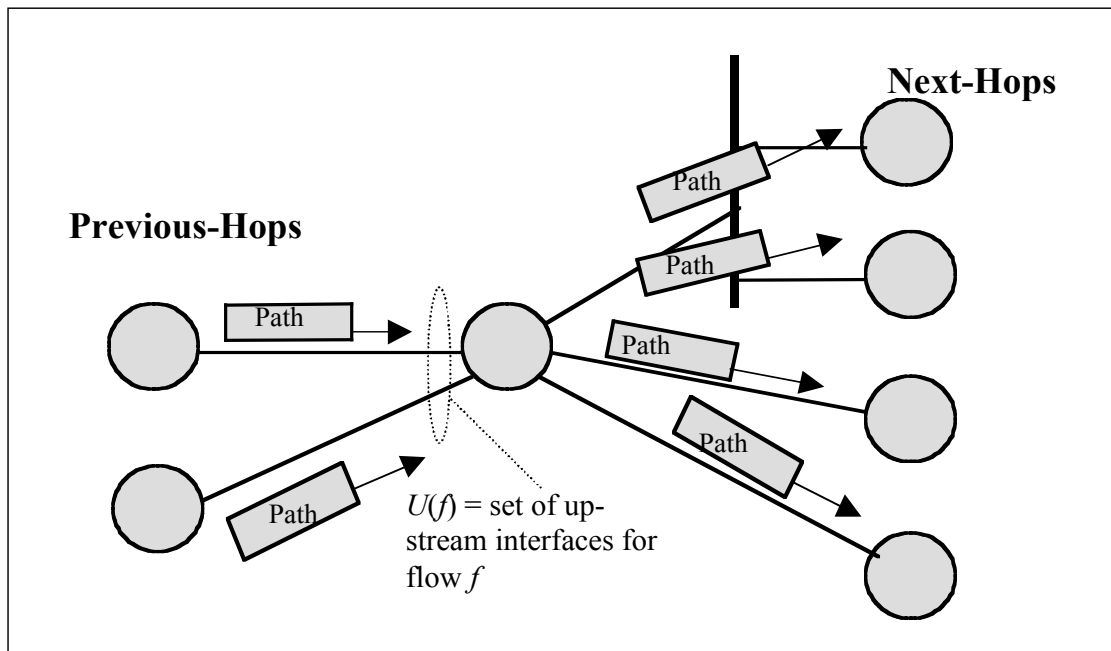
### 5.2.6 Other Messages

The other standard RSVP messages (such as PathTear, PathErr, and ResvErr) are unchanged in dRSVP.

## 5.3 Dynamic RSVP Message Processing

### 5.3.1 Path processing

The Path messages are sent from each sender along a path to its receiver(s). If the destination address is multicast, the Path messages will be sent along a multicast tree to all destinations that are participating in the session.

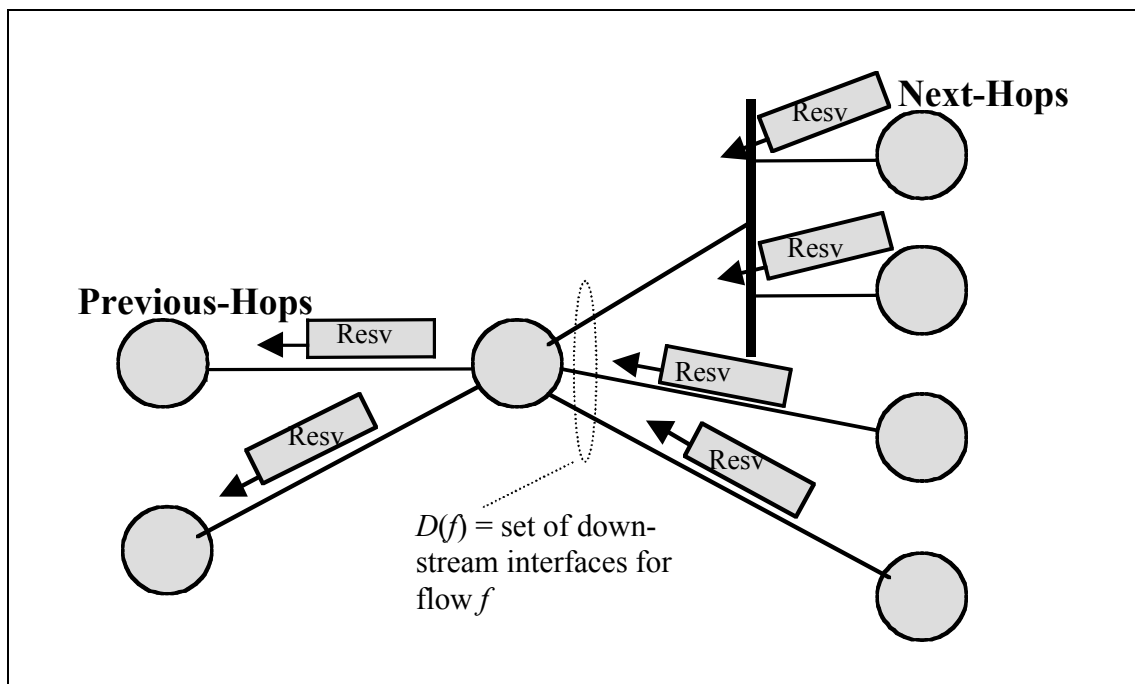


**Figure 5-3. Path Messages**

Each node that receives a Path message records the information that is obtained from the Path message (see Figure 5-3). In our implementation (based on the ISI implementation) this information is stored in a Path State Block (PSB). PSBs are stored in a complex data structure that includes several linked lists. Each linked list is formed by linking together all

PSBs that have the same session address and port. A session's PSB also contains the previous hop information used to route Resv messages in the reverse direction. In our dRSVP implementation we expand the PSB structures of ISI's RSVP implementation to accommodate the TspecLow and TspecHigh information (These may not be a one-to-one correspondence between path state and flows, since a flow includes a filterspec which specifies the list of senders, as well as a destination address. We use the notation  $U(f)$  to denote the set of upstream interfaces for which path state exists for a given flow  $f$ . The filterspec is given in the reservation, therefore  $U(f)$  is defined only after a reservation request has been issued.)

Path messages are sent along the same route as the data. At each intermediate node, the RSVP daemon receives the message and updates the associated PSB if some of the fields in the message have changed. If no PSB for the session is available, a new one is created. In the case the PSB is changed or created, new copies of the message are created and are sent downstream toward receivers. In addition, a timer is set to cause Path messages to be periodically sent downstream.



**Figure 5-4. Resv Messages**

### 5.3.2 Processing Reservation Messages

Resv messages are initiated by receivers and sent upstream. Receipt of a Resv message causes an admission control test (described below) to be performed. As with RSVP, if admission control fails, ResvTear messages are sent back towards receivers. If admission control passes, state information is saved, and a bandwidth allocation algorithm (also described below) is executed.

In our implementation, the RSVP daemon saves the reservation state information in a Reservation State Block (RSB). Like PSBs, RSBs are linked together for each destination and a hash table is used to find a pointer to the beginning of each linked list. For dRSVP, the RSB was expanded to include fields to save the second flowspec and the mspec associated with a flow or session.

We use the notation  $s_h(f,i)$  and  $s_l(f,i)$  to denote the high and low flowspecs received for flow  $f$  from downstream interface  $i$ . (The “ $s$ ” stands for “spec”; the “ $h$ ” and “ $l$ ” stand for “high” and “low”) We use the notation  $a_{dn}(f,i)$  to denote the mspec value received for flow  $f$  from downstream interface  $i$ , and we use the notation  $a(f,i)$  to denote the results of the bandwidth allocation algorithm performed at the current node for flow  $f$  on downstream interface  $i$ . (The “ $a$ ” stands for “allocation”.)

Resv messages are sent to previous hop(s) from which matching Path messages have previously arrived (as indicated by a PSB). As shown in Figure 5-4, RSVP forwards each Resv message to all previous hop (PHOP) nodes for which it has matching Path state (Path state information is stored in PSB data structure). Resv messages are forwarded to previous hops whenever a Resv message for the flow received from downstream results in a change in reservation state, or whenever a reservation refresh timer expires. It is important to note that, in order to reduce overhead, we *do not* send Resv messages as a result of other events that would affect the bandwidth allocation for this flow, such as changing link bandwidth or addition, deletion, or modification of other flows. Rather, we wait for the next refresh time before recomputing the bandwidth allocation for this flow and sending Resv messages upstream.

Unicast flows have one RSB entry for each flow. For multicast flows, the number of RSB entries may be more than one, and it is equal to the number of NHOPs that are on Paths to multicast receivers participating in a session. We denote the set of downstream interfaces for which reservation state exists for flow  $f$  as  $D(f)$ .

Unicast reservations are independent from each other at each node and are processed independently. However, multicast flows using the same destination address and the same filterspec are merged together into a single Resv message sent on each upstream interface.

When sending Resv messages to upstream nodes, we need to report the following values in the Resv message:

$m_r$  The mspec value. (This will be stored in  $a_{dn}(f,i)$  on the upstream node)

$f_h$  FLOWSPEC\_HIGH. (Stored in  $s_h(f,i)$  on the upstream node)

$f_l$  FLOWSPEC\_LOW. (Stored in  $s_l(f,i)$  on the upstream node)

The receiver mspec value  $m_r$  for flow  $f$  on any upstream interface is intended to indicate to the upstream node the existence of any downstream bottleneck, which could be caused by this node or by someone downstream from this node. The value of  $m_r$  is computed as follows:

$$m_r = \min[a(f), a_{dn}(f)]$$

where  $a(f)$  is the minimum of the allocations (computed by the bandwidth allocation algorithm described below) on all of the downstream interfaces for flow  $f$ :

$$a(f) = \min_{i \in D(f)} [a(f,i)]$$

and  $a_{dn}(f)$  is the minimum bandwidth allocated for  $f$  on any downstream interface:

$$a_{dn}(f) = \min_{i \in D(f)} [a_{dn}(f,i)].$$

Since  $m_r$ ,  $a$ , and  $a_{dn}$  are flowspecs, which are vectors rather than scalar values, we need to define how we compute “minimum” and “maximum” values. As discussed in [6], the way flowspecs are ordered and merged depends on the service model. For the controlled-load service, flowspecs contain rate  $r$ , bucket depth  $b$ , peak rate  $p$ , minimum policed unit  $m$ , and maximum packet size  $M$ . For two flowspecs  $f_1 = (r_1, b_1, p_1, m_1, M_1)$  and  $f_2 = (r_2, b_2, p_2, m_2, M_2)$  we define the maximum and minimum<sup>8</sup> of  $f_1$  and  $f_2$  as:

$$\begin{aligned} \max(f_1, f_2) &= (\max(r_1, r_2), \max(b_1, b_2), \min(m_1, m_2), \max(M_1, M_2)) \\ \min(f_1, f_2) &= (\min(r_1, r_2), \min(b_1, b_2), \max(m_1, m_2), \min(M_1, M_2)) \end{aligned}$$

The reason we use the minimum rather than the maximum when merging allocation values to compute  $m_r$  is that we assume the senders are going to limit transmission rates so that they can stay within all reservations established. Therefore, we set the mspec to indicate the smallest reservation in effect downstream. This is different from the standard RSVP notion that different receivers can reserve different reservations, and upstream nodes will merge these reservations to accommodate the *largest* reservation needed downstream. However, note that this computation is for the allocation within the range, not for the upper end of the range. For merging the upper and lower ends of the ranges, we maintain the standard RSVP logic.

---

<sup>8</sup> [6] and [14] use the terms Least Upper Bound (LUB) and Greatest Lower Bound (GLB) rather than “maximum” and “minimum”.

The flowspec range reported in the reservation request sent to upstream nodes should be set wide enough to encompass all reservation requests we have received from any of the downstream nodes:

$$f_u = \max_{j \in D(f)} [s_h(f, j)]$$

$$f_l = \min_{j \in D(f)} [s_l(f, j)]$$

### 5.3.3 Processing for ResvNotify Messages

When Resv messages reach a sender, the sender initiates a ResvNotify that is sent towards all destinations for that flow. The function of this message is to allow each node to learn the maximum bandwidth that is allocated for the flow on all upstream nodes. Receipt of ResvNotify messages cause the sender mspec (smspec) value to be stored in the RSB. We use the notation  $a_{up}(f, i)$  to denote the last smspec value received for flow  $f$  from upstream interface  $i$ .

When sending ResvNotify messages to downstream nodes, we need to report the smspec value,  $m_s$ . The value  $m_s$  reported for flow  $f$  on downstream interface  $i \in D(f)$  is intended to indicate to the downstream node the existence of any upstream bottleneck, which could be caused by this node or by another node upstream from this node. Let  $a_{up}(f)$  denote the worst case rate we expect to receive from any upstream transmitter for a given flow  $f$ , and let  $a(f, i)$  denote the allocation we have computed for this flow on downstream interface  $i$ . Then the value of  $m_s$  to be reported downstream on interface  $i$  is computed as follows<sup>9</sup>:

$$m_s = \min \left[ a(f, i), a_{up}(f) \right]$$

The computation of  $a(f, i)$  is defined by the bandwidth allocation algorithm, given below. The computation of  $a_{up}(f)$  is as follows:

$$a_{up}(f) = \max_{j \in U(f)} \left[ a_{up}(f, j) \right]$$

(Recall that  $U(f)$  denotes the set of upstream interfaces for flow  $f$ .)

## 5.4 Admission Control

When a Resv message for a new flow  $f$  is received from downstream interface  $i$ , an admission control check must be performed. The new flow should be admitted if and only if its requested bandwidth, plus the bandwidths of all the existing flows, will fit "safely" within the currently available link bandwidth. If  $F(i)$  is the set of existing (i.e. already admitted

---

<sup>9</sup> See implementation note 1 in Appendix A.

flows) on downstream interface  $i$ , and  $b(i)$  is the total current bandwidth of interface  $i$ , then we admit  $f$  on  $i$  iff:

$$\omega_n(f) + \sum_{g \in F(i)} \omega_e(g) \leq \alpha_i b(i) ,$$

where  $\omega_n$  and  $\omega_e$  are functions that determine how much bandwidth we attribute to new flows and existing flows, respectively, for admission control purposes. For example, a function that uses the low requested bandwidth plus some fraction of the requested bandwidth range would be:

$$\omega_n(f) = \gamma_n s_l(f, i) + (1 - \gamma_n) s_h(f, i) ,$$

where  $\gamma_n$  is a configuration parameter that tunes the algorithm, with  $0 \leq \gamma_n \leq 1$ . Presumably  $\gamma_n$  would be set at or close to 1, so that admission control would be based entirely or mostly on the minimum requested bandwidth of a new flow.

Similarly, for  $\omega_e$  we could use:

$$\omega_e(f) = \gamma_e s_l(f, i) + (1 - \gamma_e) s_h(f, i)$$

where  $\gamma_e$  is another configuration parameter, with  $0 \leq \gamma_e \leq 1$ .

In the admission control decision,  $\alpha_i$  controls how optimistic the admission control algorithm will be in terms of expecting that the link bandwidth will not degrade below the current value. Presumably  $\alpha_i$  would be set to 1 for a fixed speed wired link, and at a lower value for a wireless link or a variable speed wired link.

In our implementation, we set  $\gamma_e = \gamma_n = 1$ , so admission control is based solely on the low bandwidth values for existing flows and the new flow. We also set  $\alpha_i$  to 1, making the algorithm extremely optimistic (probably overly optimistic for a real system).<sup>10</sup>

An alternative admission control scheme would be to use a formula which takes into account the current observed rates for existing flows, such as:

$$\omega_e(f) = \gamma_e c(f) + (1 - \gamma_e) s_h(f, i) ,$$

where  $c(f)$  represents an estimate of the current actual rate of transmission of flow  $f$ . This alternative is not explored further here.

---

<sup>10</sup> See implementation note 2 in Appendix A.

## 5.5 Bandwidth Allocation Algorithm

The bandwidth allocation algorithm determines how much bandwidth on each downstream interface is allocated to each flow. This algorithm could, in theory, be run whenever any key parameter changes (e.g., a flow is admitted or torn down, changes to  $b(i)$  are detected for any downstream link, new values of  $a_{up}$  or  $a_{dn}$  are received from upstream or downstream links in Path or Resv messages). However, this would create considerable protocol overhead.

Therefore, we execute the bandwidth allocation algorithm only when it is time to send a Resv upstream to refresh reservation state for a given flow. Furthermore, even though the algorithm computes new bandwidth allocations for all flows, to avoid creating a cascading “storm” of Resv messages, we only send the Resv upstream for the flow being refreshed. This keeps overhead low, at the expense of delaying possible application adaptation. The sender application will only learn about reservation allocation changes at most once per refresh interval. If the interval is long with respect to the rate at which the network resources are changing, the application may over or under utilize the network, possibly resulting in non-conforming application traffic which would be subject to policing.

Before defining the bandwidth allocation algorithm, we need to describe how we aggregate flowspecs, and introduce the concept of external allocation.

### 5.5.1 Flowspec Aggregation

A given flow may have receivers on more than one downstream interface, and it is possible that we could receive different flowspecs from the different receivers. The flowspec range values are aggregated as follows:

$$s_h(f) = \max_{i \in I(f)} (s_h(f, i))$$

$$s_l(f) = \min_{i \in I(f)} (s_l(f, i))$$

### 5.5.2 External Allocation

Central to the dRSVP bandwidth allocation algorithm is the concept of "external allocation",  $a_{ext}$ . The external allocation,  $a_{ext}(f, i)$ , represents the amount we need to allocate for flow  $f$  in order for this node not to represent a bottleneck for flow  $f$  on interface  $i$ . For a given flow, the dRSVP algorithm will attempt to allocate at least as much as has been allocated elsewhere in the network. This is computed based on the allocation being reported to the node from upstream and downstream nodes.

We may have multiple senders upstream, and we want to be able to reserve enough to handle the worst case sender, so we will need to reserve as much as the largest reservation in effect upstream. This is given by:

$$a_{up}(f) = \max_{i \in U(f)} [a_{up}(f, i)] .$$

Recall that  $a_{up}(f,i)$  is the value obtained from the smspec in ResvNotify messages received for flow  $f$  on upstream interface  $i$ . If we are at the source of this flow (i.e. if we have an upstream "interface"  $j$  for this flow which is the API), then we set

$$a_{up}(f,j) = s_h(f) .$$

If no ResvNotify messages have been received for this flow on any upstream interfaces, (as will be the case when the reservation is being set up initially) then we also set:

$$a_{up}(f) = s_h(f) .$$

Similarly, we may have multiple receivers downstream. We assume that senders will adapt to a rate that can be reliably delivered to *all* receivers, so we need to reserve enough to support the smallest downstream pipe. This is given by:

$$a_{dn}(f) = \min_{i \in D(f)} [a_{dn}(f,i)]$$

Finally, in order for this node not to be a bottleneck, we need an allocation that is at least as big as the smaller of the upstream and downstream allocations. This is given by:

$$a_{ext}(f,i) = \min [a_{up}(f), a_{dn}(f)]$$

### 5.5.3 Bandwidth Allocation Algorithm

This algorithm computes the bandwidth to allocate to all flows on downstream interface  $i$ .

First, we compute some "goals" for bandwidth allocation:  $H$ ,  $A_{ext}$ , and  $L$ .

$H$  is defined as the total bandwidth that we would need in order to provide the maximum desired bandwidth for all downstream flows that have been admitted on this interface:

$$H = \sum_{f \in F(i)} s_h(f)$$

(Recall that  $F(i)$  is the set of flows that have been admitted on downstream interface  $i$ .)

$A_{ext}$  is defined as the total bandwidth we would need in order to provide at least as much bandwidth as has been reserved externally (downstream or upstream):

$$A_{ext} = \sum_{f \in F(i)} a_{ext}(f)$$

Finally,  $L$  is defined as the total bandwidth we would need in order to provide each flow the minimum required bandwidth:

$$L = \sum_{f \in F(i)} s_l(f)$$

In order to maximize network utilization, the algorithm is designed to allocate as much bandwidth as possible. We examine each of the “goals” computed above in turn, and if the goal can be met, we give away the “goal” bandwidth, and then divide up any leftover bandwidth among the flows. This is specified below (where  $b(i)$  represents the available bandwidth on interface  $i$ ):

if  $H \leq b(i)$  then

We have bandwidth to spare! Give each flow the maximum requested:

for each flow  $f$

$$\text{padding-left: 80px; } a(f,i) = s_h(f)$$

end for

else if  $A_{ext} \leq b(i)$  then

Give each flow at least as much as has been allocated upstream and downstream, and divvy up any remainder proportionately:

$$\text{padding-left: 40px; set } \beta = \frac{b(i) - A_{ext}}{H - A_{ext}}$$

for each flow  $f$

$$\text{padding-left: 80px; } a(f,i) = a_{ext}(f) + \beta[s_h(f) - a_{ext}(f)]$$

end for

else if  $L \leq b(i)$  then

Give each flow at least the minimum of its range, and divvy up any remainder proportionately:

$$\text{padding-left: 40px; set } \beta = \frac{b(i) - L}{A_{ext} - L}$$

for each flow  $f$

$$\text{padding-left: 80px; } a(f,i) = s_l(f) + \beta[a_{ext}(f) - s_l(f)]$$

end for

else

There is insufficient capacity to maintain even the minimum. In this case we must either reject (tear down) some flows<sup>11</sup>, or allow “over subscription”, in the hope that the bandwidth of the interface will soon be restored to an acceptable value:

---

<sup>11</sup> See implementation note 3 in Appendix A.

for each flow  $f$ .

$$a(f,i) = s_l(f)$$

end for

end if

Thus, if possible, we allocate each flow the maximum requested. If this is not possible, we divide up the available bandwidth among the flows, giving each its minimum, plus an equal fraction ( $\beta$ ) of its requested range. We can easily see that we allocate *all* of the available bandwidth. For example, consider the second case, where  $A_{ext} \leq b(i) < H$ . In this case the total bandwidth allocation on interface  $i$  is:

$$\begin{aligned} a_{tot} &= \sum_{f \in F(i)} a(f) = \sum_{f \in F(i)} a_{ext}(f) + \sum_{f \in F(i)} \beta [s_h(f) - a_{ext}(f)] \\ &= \sum_{f \in F(i)} a_{ext}(f) + \beta \left[ \sum_{f \in F(i)} s_h(f) - \sum_{f \in F(i)} a_{ext}(f) \right] \\ &= A_{ext} + \beta [H - A_{ext}] \\ &= A_{ext} + \frac{b(i) - A_{ext}}{H - A_{ext}} [H - A_{ext}] \\ &= b(i) \quad . \end{aligned}$$

A similar proof easily shows that the same is true in the case where  $L \leq b(i) < A_{ext}$ .

#### 5.5.4 Analysis of the Bandwidth Allocation Algorithm

On first examination, it may seem wasteful or overly aggressive to allocate the entire link bandwidth. However, we believe that a closer consideration shows that the algorithm will cause applications to fully utilize bottleneck links, and to adapt when any changes occur in bottlenecks, but will not cause application adaptation when changes occur at other, non-bottleneck points in the network.

First, consider the case where downstream interface  $i$  is a bottleneck link, that is,  $L \leq b(i) < A_{ext}$ . In this case, any variations in the link bandwidth will result in changes in allocation for all flows over this interface. These will be reported upstream and downstream (not immediately, but at an interval based on the reservation refresh timer). This will cause all application flows that traverse this interface to receive a new allocation, and adapt within their ranges to fully utilize the bottleneck link. Similarly, new flows being admitted on this

interface, or existing flows being torn down, will cause application flows that traverse this interface to adapt to fully utilize the link.

Next, consider the case where  $A_{ext} \leq b(i) < H$ . In this case, we do not have sufficient bandwidth to fully satisfy the “high” requirement for every flow on interface  $i$ , but we do have enough to give each flow at least as much bandwidth as has been allocated externally. Therefore, we are not, in this case, a bottleneck for any flow traversing this link. This means that, as long as we remain in this case, changes in the link bandwidth and flows being added or dropped on this interface will not result in applications having to adapt.

Another question that may arise is why we would want to compute an “allocation” for a flow  $f$  that is larger than can be utilized due to bottlenecks elsewhere in the network. In other words, why, in the case where  $A_{ext} \leq b(i) < H$ , do we compute an  $a(f,i)$  which is greater than  $a_{ext}(f)$ ? In fact, to minimize processing overhead associated with modifying CBQ flows, we never actually reserve more than  $a_{ext}(f)$  for flow  $f$ . However, if a bottleneck for flow  $f$  is removed elsewhere in the network, we want the former bottleneck node to immediately increase the allocation. For example, assume that a bottleneck exists downstream. Our computed value of  $a(f,i)$  is reported downstream in the ResvNotify messages. As these messages flow downstream, each node takes the minimum of its computed allocation value and the values reported upstream. This allows a downstream bottleneck node to know when the bottleneck there is relieved and how much the nodes upstream from it are actually capable of reserving. Thus, when the bottleneck is relieved, the former bottleneck node knows how much bandwidth each flow can utilize before it is limited by upstream nodes, and this allows it to immediately make the maximal reservation for all flows. Similarly, the Resv messages flowing upstream allow upstream nodes that had been bottlenecks to immediately respond when the bottleneck is relieved.

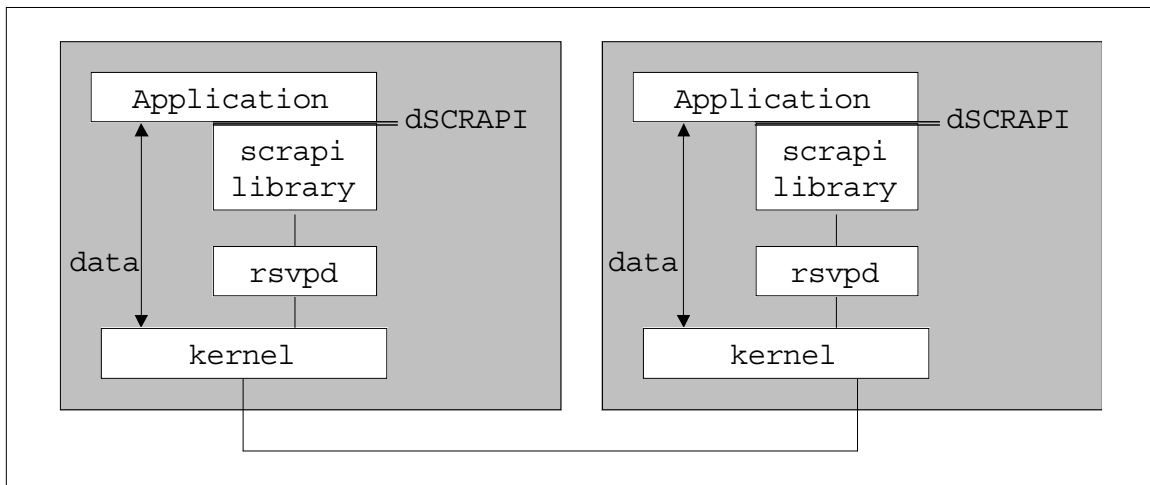
## 5.6 dRSVP Application Programming Interface

The dRSVP Application Programming Interface (dSCRAPI) is based on the SCRAPI interface provided with ISI’s RSVP implementation [28]. The key dSCRAPI calls are shown in Table 5-1.

**Table 5-1. dSCRAPI Routines**

Routine	Purpose	Notes
scrapi_sender_bwrange()	Registers an application as a data sender and specifies bandwidth range that sender can operate in.	Replaces scrapi_sender()
scrapi_receiver()	Registers an application as a receiver, and attempts to reserve bandwidth back to sender.	Calling sequence unchanged, internals modified to deal with bandwidth ranges
scrapi_getfd(), scrapi_dispatch()	Used to cause asynchronous "upcalls" to provide feedback when network conditions have changed.	Unchanged
scrapi_get_bw()	Gets status of reservation (pending, good, or bad), and currently available bandwidth.	Replaces scrapi_get_status()

The dSCRAPI interfaces are provided by a set of library routines, which are linked in to the application and communicate with the dRSVP daemon by means of a socket, as shown in Figure 5-5.



**Figure 5-5. dSCRAPI Relationship to Other Components**

In addition to new routines added in dSCRAPI, the original SCRAPI functions continue to be supported, with unchanged calling sequences. dSCRAPI maps the old functions, which do not support the notion of bandwidth ranges, into the new paradigm. For example, calling

```
scrapi_sender( ..., bandwidth, ... )
```

results in a call to

```
scrap_sender_bwrap ( ..., bw_low, bw_high, ... )
```

with

```
bw_low = bw_high = bandwidth.
```

In this way, applications written to work with SCRAPI and RSVP will work unchanged with dSCRAPI and dRSVP, simply by re-linking the dSCRAPI library in place of the SCRAPI library.

## Section 6

# Test and Evaluation

## 6.1 Test Environment

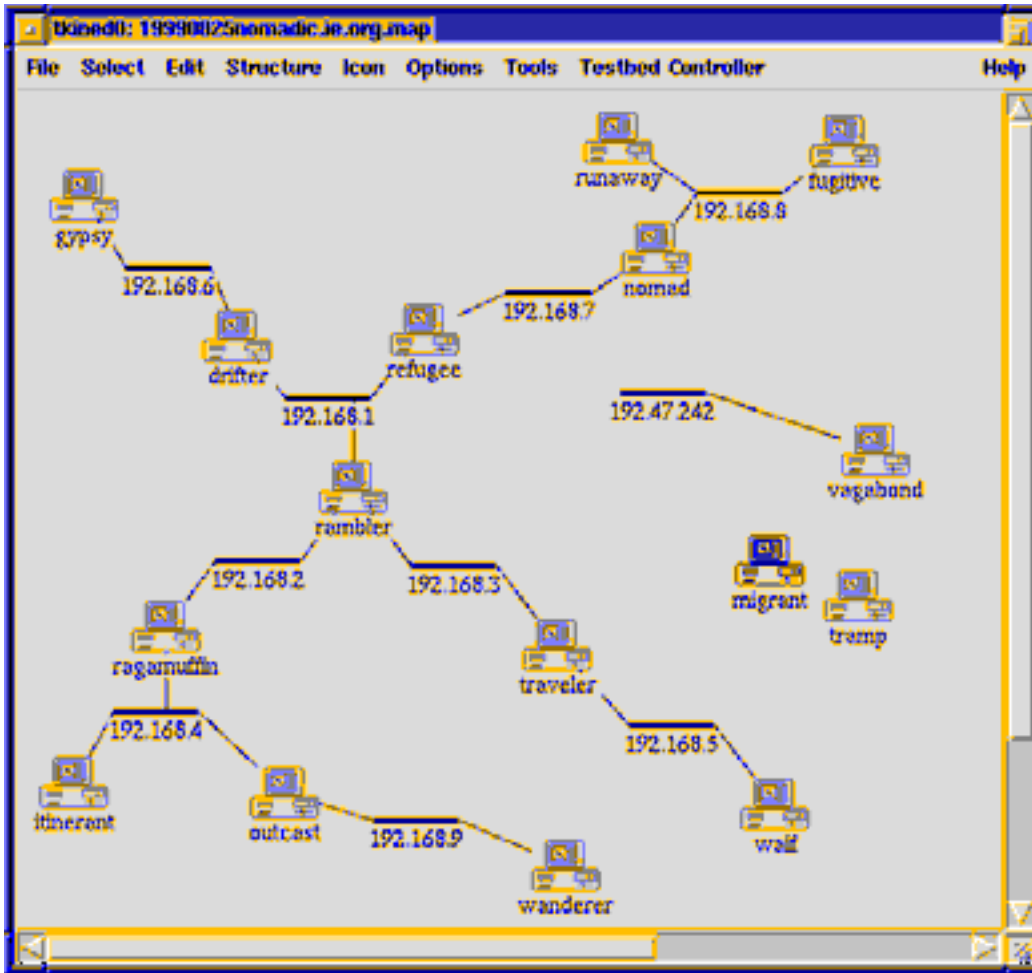
In order to test and evaluate our implementation of the adaptive QoS protocol described above, we have created a testbed in which we can vary the bandwidth of any link within a network of routers. These routers can be configured in a variety of complex network topologies. The routers are PCs, running the FreeBSD operating system, with CPU speeds from 90 to 300 MegaHertz (MHz). We implemented the dRSVP protocol described in this paper by modifying a version of ISI's rsvpd (RSVP daemon) code [27] together with CBQ code from the Alternate Queueing (ALTQ) package [16], [17].

To accomplish varying link speeds, we created a centralized testbed controller application, which was built using the Tkined/Scotty package [29]. The testbed controller provides a GUI as well as a scripting facility from which we can set the speed of any of the links in the testbed. The testbed controller sends commands to a bandwidth manager daemon resident on each router in the testbed, which then implements the link speed change command by interacting, via rsvpd, with the ALTQ package. The link speed change is effected by modifying the queue service parameters used in CBQ. By examining the link traffic, we verified that this strategy is an effective way to vary the effective link speed seen by the network layer. The only problem we have observed with this technique is that the interface bandwidth actually consumed by CBQ is somewhat sensitive to packet size. Flows with small packets tend to be under-served; that is, they actually receive less bandwidth than specified.

Figure 6-1 is a screenshot of the Testbed Controller application interface. The screenshot shows the routers and links of the testbed, in a configuration we use for running demonstrations of the dRSVP protocol.

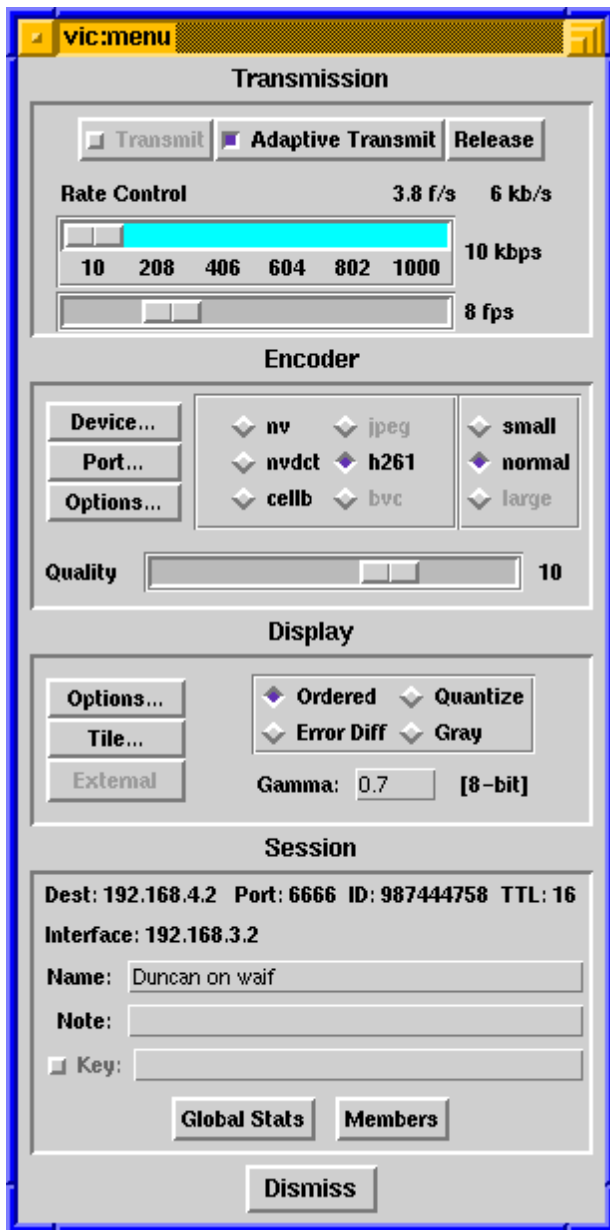
## 6.2 Test Applications

In order to verify the dynamic QoS mechanisms added to rsvpd, and the adaptive QoS API, we have created a set of adaptive applications. First, we modified the rsvpd-aware multicast data generator and receiver applications mgen and drec, created by Naval Research Labs (NRL). These applications provide a nice tool for generating test traffic for laboratory experiments; however they do not provide much insight into the value of dynamic QoS for a realistic application. For that, we selected the streaming video and applications vic and vat, which were created by LBL, and modified to be rsvpd-aware by ISI. We applied some rather extensive modifications to these applications to use the new dSCRAPI API described in Section 5.



**Figure 6-1. Testbed Controller Application**

Figure 6-2 shows a screenshot of the user interface of our modified version of the video tool vic. A key feature of this interface is the “Rate Control” slider. In the normal version of vic, this slider enables the user to control the maximum rate, in Kbps, at which video data is transmitted. When a new transmission rate is selected, the application will adjust the frame rate as necessary to stay within this limit. Those familiar with the standard version of this application will notice that we have added an “Adaptive Transmit” button alongside the standard “Transmit” button. The “Transmit” button invokes normal vic operation, in which the transmission rate slider is controlled manually by the user. The new “Adaptive Transmit” button invokes the new adaptive QoS behavior. In this mode, the transmit rate slider is no longer controllable by the user. Instead, the transmission rate is set



**Figure 6-2. Screenshot of “vic” Video Application**

to the rate allocated by the network in response to reservation requests from receivers<sup>12</sup>.

The end points of the transmission rate slider (10 Kbps and 1 Mbps in Figure 6-1) determine the QoS range that will be requested; this range is configurable by the user at application start-up time from the command line or a configuration file. In “Adaptive Transmit” mode, our version of vic automatically adapts to changes in transmission rate signaled by the network (via the dSCRAPI API) by adjusting the frame rate. The slider on the GUI moves to show the current transmission rate, and the color of the slider shows dRSVP reservation status – pending, reserved, or failed.

Image quality and other video parameters (encoding format, color versus monochrome, etc.) can be manually adjusted by the user, using the sliders and buttons available in the GUI. These parameters will affect the frame rate that can be achieved at a given transmission rate. For example, at slow transmission rates reasonable frame rates can be obtained by decreasing image quality. While our implementation’s QoS adaptation mechanism consists simply of adjusting the frame rate, a more sophisticated application could use some heuristics to choose a complete set of transmission parameters appropriate to the network conditions.

<sup>12</sup> To be precise, in “adaptive transmit” mode the application transmits at a rate that can be reliably delivered to all receivers who have issued reservations. Nodes can join the video multicast group without issuing reservations, in which case they will receive best effort service and will not impact the sender's transmission rate.

The modifications we made to vic illustrate the suitability of the adaptive QoS paradigm for programming streaming applications. The source knows the range of network bandwidth it is capable of utilizing, the actual transmission rate is selected dynamically according to current network conditions, and the application adjusts its transmission rate to match what the network can currently support.

Figure 6-3 shows a screenshot from our modified version of the audio tool vat. Again, we have added an “Adaptive Transmit” button, which causes the application to transmit at a rate that can be reliably received by all destinations. In this case, the application adapts to changes in network bandwidth by changing the encoding algorithm to one which requires no more than the available bandwidth signaled by dRSVP via the dSCRAPI API. The minimum bandwidth is 9 Kbps in the gsm4 mode (GSM encoding with 4 audio samples per IP datagram). The maximum bandwidth is 78 Kbps in pcm1 mode (PCM encoding, with one sample per IP datagram).

While our version of vat worked well, and provided a good demonstration of the value of dynamic QoS, results with vat were less impressive than with vic, and quality was not always excellent even with reservations in place. Part of the reason for this is that the controlled load service model is not as successful for audio as it is for video. Another reason is that the CBQ implementation underlying the QoS mechanisms in our routers is sensitive to packet size, and under-serves queues with many small packets, as is the case with audio flows. A possible avenue for further experimentation

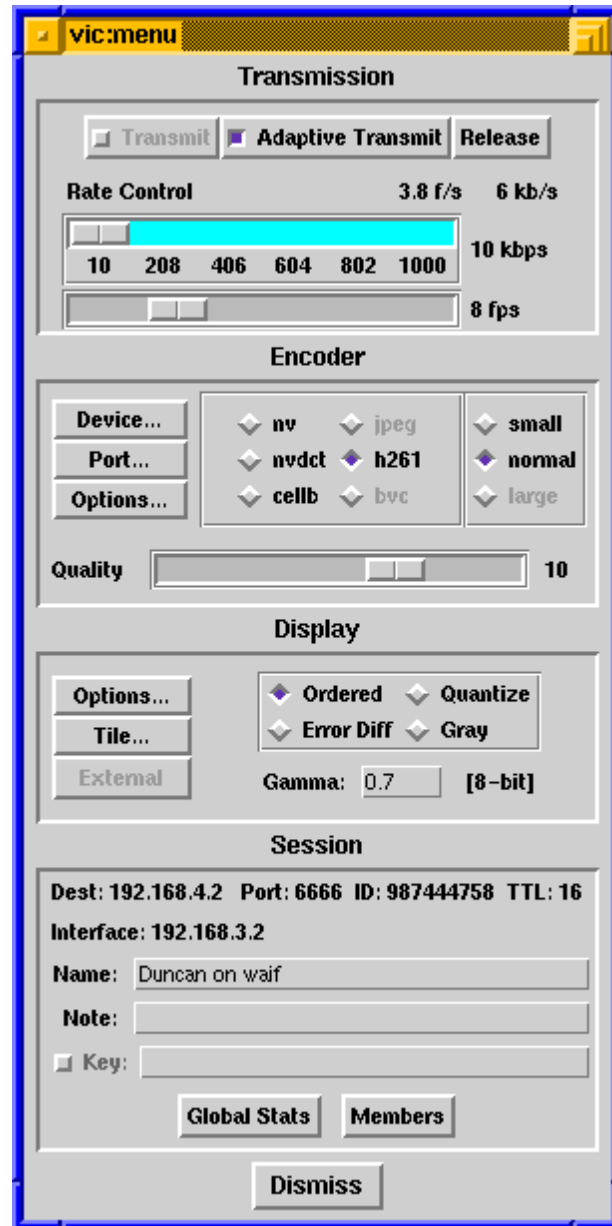


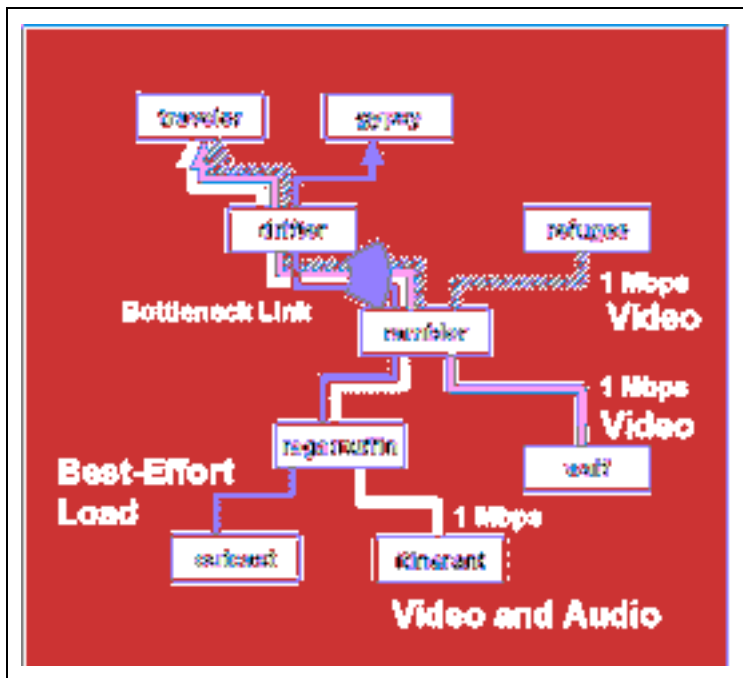
Figure 6-3. Screenshot of “vat” Audio Application

would be to experiment with an audio tool such as vat, enabled with dynamic QoS together with an implementation of the Guaranteed-Load service model and a more refined CBQ implementation.

### 6.3 Testbed

Figure 6-4 shows a testbed configuration that we use to demonstrate the benefits of the adaptive QoS approach. Using vic and vat, we generate adaptive video and audio flows through the network, and using mgen and drec, we load the network with best-effort traffic, as shown by the arrows in the figure. With this type of configuration, we can qualitatively demonstrate the benefits of our adaptive QoS approach. Our demonstration clearly shows two advantages to this approach. First, we can show the ability of applications to make use of the dRSVP capabilities to adapt to changing resource levels (link bandwidth) within the network. Second, we can show the ability of applications to make use of dRSVP to effectively share resources.

We can also use a configuration such as this for testing and demonstrating the features of the



**Figure 6-4. Demonstration Scenario**

dRSVP protocol. For example, in this configuration we have a bottleneck on the link from rambler to drifter, which results in sharing the bandwidth of this link among the video and audio flows being generated on itinerant, waif, and refugee. If we then subscribe to the waif-generated video flow on gypsy, and create an even tighter bottleneck on the link from drifter to gypsy, we can show how the waif-generated video flow scales back, releasing bandwidth on

the remaining links for use by the other flows.

Our next task is to utilize the testbed to perform quantitative experiments on the benefits and costs (CPU and network overhead) of our approach. This work is currently underway.



## Section 7

# Conclusions

With our testbed, applications, and dRSVP implementation, we are able to demonstrate a complete system in which QoS support is maintained, even while link bandwidths vary within the network. Our experience in developing this capability has convinced us that an adaptive QoS approach is both feasible and potentially valuable. We are in the process of gathering quantitative results on various aspects of our implementation, for example we plan to gather data on protocol overhead under various conditions to make possible analysis of scalability.

Many interesting possibilities remain open for investigation. One possible area would be to apply the notion of reservation ranges to other service models, for example Guaranteed QoS [25]. Another area for further study is the interaction between adaptive QoS and a variety of different link layers, in particular a shared media link layer with a subnet bandwidth manager for link layer resource management. We would like also like to investigate is how an adaptive QoS approach such as that proposed in this paper would fit into an overall intserv/diffserv environment. Other possible areas for further research include integrating an adaptive QoS approach with a (separate) QoS routing solution, and applying the notion of bandwidth ranges together with a lightweight QoS signaling mechanism such as INSIGNIA, in a mobile ad hoc network environment. Our hope is that the concepts and experience documented in this paper will encourage further research into these areas.

## List of References

- [1] M. Stemm, R. Katz; *Vertical Handoffs in Wireless Overlay Networks*; ACM Mobile Networking (MONET), Special Issue on Mobile Networking in the Internet; Winter 1998.
- [2] S. Lu, V. Bharghavan; *Adaptive Resource Management Algorithms for Indoor Mobile Computing Environments*; Proceedings of ACM SIGCOMM '96, Univ. of Illinois at Urbana-Champaign, Stanford, CA; August 1996.
- [3] S. Lu, K. Lee, V. Bharghavan; *Adaptive Service in Mobile Computing Environments*; from "*Building QoS into Distributed Systems*"; Campbell and Nharstedt (editors); Chapman & Hall; 1997.
- [4] O. Angin, A. Campbell, M. Kounavis, R. Liao, *The Mobeware Toolkit: Programmable Support for Adaptive Mobile Computing*, IEEE Personal Communications Magazine, August 1988.
- [5] S. Lee and A. Campbell, *INSIGNIA: In-band Signaling Support for QOS in Mobile Ad Hoc Networks*, Proc of 5th International Workshop on Mobile Multimedia Communications (MoMuC,98), Berlin, Germany, October 1998.
- [6] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin. *Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification*, IETF RFC 2205, September 1997.
- [7] R. Braden, L. Zhang; *Resource ReSerVation Protocol (RSVP) -- Version 1 Message Processing Rules*, IETF RFC 2209, September 1997.
- [8] R. Bagrodia, W. Chu, L. Kleinrock, C. Popek, *Vision, Issues, and Architecture for Nomadic Computing [and Communications]*, IEEE Personal Communications Volume: 2 6 , Page(s): 14 –27, December 1995.
- [9] ITU Telecommunication Standardization Sector of ITU, *V.90 - A Digital Modem and Analogue Modem Pair for Use on the Public Switched Telephone Network (PSTN) at Data Signalling Rates of up to 56000 Bit/S Downstream and up to 33600 Bit/S Upstream*, Section 9.6 – Rate Renegotiation, September 1998.
- [10] Scientific Research Corp, *Turbolink™ White Paper*, January 2000.  
<http://www.scires.com/turbo.htm>
- [11] D. Beyer, T. Frivold, J. Hight, M. Lewis, *API Framework for Internet Radios*, September 1998. [ftp://ftp.rooftop.com/pub/apis/api\\_framework.pdf](ftp://ftp.rooftop.com/pub/apis/api_framework.pdf)

- [12] Yavatkar, Hoffman, Bernet, Baker, Speer, *SBM (Subnet Bandwidth Manager): A Protocol for RSVP-based Admission Control over IEEE 802-style Networks*; IETF Internet Draft (work in progress); January 2000.  
<http://www.ietf.cnri.reston.va.us/internet-drafts/draft-ietf-issll-is802-sbm-10.txt>
- [13] E. Horlait, M. Bouyer, *CLEP (Controlled Load Ethernet Protocol): Bandwidth Management and Reservation Protocol for Shared Media*, Internet Draft (work in progress); draft-horlait-clep-00.txt, July 1999.
- [14] J. Wroclawski, *Specification of the Controlled-Load Network Element Service*, Internet Engineering Task Force Request For Comments Number 2211, September 1997.
- [15] S. Floyd, V. Jacobson; *Link-Sharing and Resource Management Modules for Packet Networks*; IEEE/ACM Transactions on Networking, Vol. 3, No. 4, pp 365-386; August 1995.
- [16] K. Cho, *A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX Based Routers*, Proceedings of USENIX 1998 Annual Technical Conference, New Orleans LA, June 1998. [www.csl.sony.co.jp/~kjc/kjc/papers.html](http://www.csl.sony.co.jp/~kjc/kjc/papers.html)
- [17] K. Cho, *Managing Traffic with ALTQ*; Proceedings of USENIX, 1999 Annual Technical Conference: FREENIX Track, Monterey CA; June 1999.  
[www.csl.sony.co.jp/~kjc/kjc/papers.html](http://www.csl.sony.co.jp/~kjc/kjc/papers.html)
- [18] S. Chen, K. Nahrstedt, *Distributed Quality-of-Service Routing in Ad Hoc Networks*, IEEE Journal on Selected Areas in Communications, Vol 17, No. 8, August 1999.
- [19] C. Lin, J. Liu, *QoS Routing in Ad Hoc Wireless Networks*, IEEE Journal on Selected Areas in Communications, Vol 17, No. 8, August 1999.
- [20] R. Sivakumar, P. Sinha, V. Bharghavan *CEDAR: A Core-Extraction Distributed Ad Hoc Routing Algorithm*, IEEE Journal on Selected Areas in Communications, Vol 17, No. 8, August 1999.
- [21] A. Talukdar, B. Badrinath, Rutgers University, and A. Acharya, C&C Research Labs, *On Accommodating Mobile Hosts in an Integrated Services Packet Network*, NEC USA, Princeton, NJ Proceedings of INFOCOM '97, Kobe, Japan, April 1997.
- [22] A. Talukdar, B. Badrinath, *MRSVP: A Reservation Protocol for an Integrated Services Packet Network with Mobile Hosts*, Department of Computer Science Technical Report DCS-TR-337, Rutgers University, July 1997.

- [23] D. Clark, S. Shenker; *Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism*; Proceedings of IEEE SIGCOMM; 1992.
- [24] A. Charny, D. Clark, R. Jain, *Congestion Control with Explicit Rate*, Indication Proceedings, ICC, June 1995.
- [25] S. Shenker, C. Partridge, R. Guerin, *Specification of Guaranteed Quality of Service*, Internet Engineering Task Force Request For Comments Number 2212, September 1997.
- [26] G. Bianchi, A. Campbell, R. Liao, *On Utility-Fair Adaptive Services in Wireless Networks*, Proceedings IEEE Sixth International Workshop on QoS, pp 256-267, 1998.
- [27] ISI, The RSVP Implementation developed by the University of Southern California (USC) Information Sciences Institute (ISI). <http://www.isi.edu/div7/rsvp/rsvp.html>
- [28] B. Lindell, ISI; *SCRAPI – A Simple ‘Bare Bones’ API for RSVP*; Work in progress (draft-lindell-rsvp-scrapi-00.txt), August 10, 1998.  
<http://www.isi.edu/rsvp/DOCUMENTS/draft-lindell-rsvp-scrapi-02.txt>
- [29] J. Schönwälder, H. Langendörfer; *Tcl Extensions for Network Management Applications*, Proceedings of the 3rd Tcl/Tk Workshop; Toronto (Canada); July 1995.  
<http://wwwhome.cs.utwente.nl/~schoenw/scotty/>
- [30] R. Braden, D. Clark , and S. Shenker, *Integrated Services in the Internet Architecture*, Internet Engineering Task Force, Request for Comments 1633, June 1994

## Appendix

# Implementation Notes

### Note 1.

The proper approach for storing smspec is to create a new data structure such as the RSB linked list. This will allow aggregation of smspecs to be performed when there is a need to compute allocated bandwidth for a flow or when a new ResvNotify needs to be generated for transmission to downstream nodes. However, to expedite implementation, we simply place the value of the smspec in the RSB data structure. The only difficulty with this approach is that smspecs can be received from several sources and their value can fluctuate. We are interested in identifying the maximum of the smspec that are received from upstream nodes. If a received smspec is higher than the one in the RSB, the aggregated smspec will be set to the LUB of the existing and the received smspec. However, if the SMSPEC in a received ResvNotify is less than the smspec in the RSB, we don't know whether the received value is the new maximum value or not. To handle this, a timer parameter was introduced to time out old SMSPECs when newly received smspecs are lower than the smspec in the RSB. A summary of the procedure for updating smspecs in the RSB is as follows. Upon receiving a ResvNotify, perform the following operation for each RSB that is associated with the flow:

If the timer associated with RSB components is not expired do the following

If the incoming value is higher than the value in the RSB component

Reinitialize the timer

Set smspec in the RSB component to the LUB of received smspec and smspec in the RSB component

Else (the timer has expired)

-Set smspec in the RSB component to the received smspec

Reinitialize the timer

End if.

### Note 2.

In our implementation we do not have a separate admission control check. When we receive a Resv, either for an existing flow or a new flow, we run the bandwidth allocation algorithm described below. If there is insufficient bandwidth for even the minimum of all flows, we fail the flow for which the Resv message was received. The net effect is that we do perform

admission control as described above with  $\gamma_e=\gamma_n=\alpha_i=1$ . It remains for future work to add the formulas above so we could experiment with different parameter settings.

**Note 3.**

When the bandwidth allocation algorithm is unable to provide all flows the minimum bandwidth requested, our implementation will tear down the flow for which the reservation state is being refreshed. We may wish, in the future, to experiment with alternative heuristics, for example tearing down flows that are not using their minimum allocation.

