

Dynamic Bandwidth Management and Adaptive Applications for a Variable Bandwidth Wireless Environment

Mohammad Mirhakkak, *Member, IEEE*, Nancy Schult, *Associate Member, IEEE*, and Duncan Thomson

Abstract—This article describes an approach for providing dynamic quality of service (QoS) support in a variable bandwidth network, which may include wireless links and mobile nodes. The dynamic QoS approach centers on the notion of providing QoS support at some point within a range requested by applications. To utilize dynamic QoS, applications must be capable of adapting to the level of QoS provided by the network, which may vary during the course of a connection. To demonstrate and evaluate the dynamic QoS concept, we have implemented a new protocol called dynamic resource reservation protocol (dRSVP) and a new QoS application program interface (API). The paper describes this new protocol and API and also briefly discusses our experience with adaptive streaming video and audio applications that work with the new protocol in a testbed network, including wireless local area network connectivity and wireless link connectivity emulated over wired ethernet. Qualitative and quantitative assessments of the dynamic RSVP protocol are provided.

Index Terms—Adaptive applications, Internet protocol, quality of service (QoS), resource allocation, resource reservation setup protocol (RSVP), wireless communications.

I. INTRODUCTION

PROVIDING support for nomadic computing presents a number of challenges. This support has been summarized [1] 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.”

Nomadic users can be expected to encounter networks with wireless links and moving nodes. This network environment can include variable link characteristics and connectivity changes that result in variations in available bandwidth. In this paper, we discuss variable bandwidth in nomadic networks and some issues involved in providing quality of service (QoS) support in this environment. We define an approach for providing QoS support in a variable network environment in which QoS is adaptive within a range requested by an application. With this QoS model, applications must be capable of adapting to the level of QoS provided by the network, which may vary during the course of a connection. We present a new protocol called dynamic resource reservation protocol (dRSVP) that supports this paradigm and is the primary focus of this paper. Implications

for adaptive applications are also discussed. Finally, we briefly discuss our experience with adaptive streaming video and audio applications that work with the new protocol in a testbed network.

II. VARIABLE BANDWIDTH IN NOMADIC NETWORKS

In contrast to the static links used in traditional networks, wireless links are subject to variations in transmission quality due to factors such as interference and fading, which cause changes in transmission quality. If the lower layers do not detect or respond to these changes, the network layer sees an increase in lost or corrupted packets. This makes it difficult to apply network layer QoS mechanisms, which have been designed mainly to deal with congestion loss and network layer queuing effects, rather than packet loss due to link errors. Therefore, we believe that variations in transmission quality are best addressed within the physical or link layers, which can react in several ways. Possibilities include dynamic changes in modulation, automatic repeat-request, and adaptive forward error correction mechanisms. In general, the techniques employed within the link and physical layer will trade off link throughput in order to maintain low error rate, creating variable bandwidth as seen from the network layer.

Another source of variable bandwidth in nomadic networks is node movement, which has several consequences. First, it exacerbates the problem of variable link characteristics, as nodes move in and out of areas of good signal strength. Second, nodes may have to switch to different media as they move in and out of coverage. A “vertical handoff” approach has been described [2] 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. Again, this illustrates the need for QoS mechanisms to deal with variable 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. However, in a more general case of a mobile *ad-hoc* network, intermediate systems (routers) also move, resulting in relatively rapid changes in network topology. This makes the general routing problem difficult and QoS-aware routing extremely difficult. It also means that end-to-end bandwidth can change even when individual links remain stable, as topology changes can result in a new route through the network that traverses links with different available resources.

Manuscript received December 2000; revised June 2001.

The authors are with the MITRE Corporation, McLean, VA 22102 USA (e-mail: mmirhakk@mitre.org; nschult@mitre.org; duncant@mitre.org).

Publisher Item Identifier S 0733-8716(01)08477-3.

III. ISSUES WITH PROVIDING QoS SUPPORT IN A VARIABLE BANDWIDTH ENVIRONMENT

As mentioned above, a solution for providing QoS support in nomadic networks must work in the face of topology changes (either the constrained case of mobile end systems or the more general case of a mobile *ad-hoc* network). A QoS solution for these environments must also be capable of handling variations in bandwidth, both on individual links and end-to-end. In this section, we discuss several issues that arise when providing QoS support in this dynamic environment, and we describe our approach to addressing these issues.

Two complementary approaches have been proposed for providing QoS support in traditional networks. A resource reservation-based approach provides QoS support by performing admission control and reserving resources for flows or connections on an end-to-end basis. A differentiated services approach provides QoS support by providing individual packets with different per-hop behaviors at a given node, depending on type of service markings on individual packets. A resource reservation-based approach is problematic in a variable bandwidth environment. If available resources change after admission control has been performed, the network may not be able to meet commitments for flows that have been successfully admitted. Nevertheless, we believe that a resource-reservation based approach is important for applications that need a per-flow, end-to-end QoS solution, and our work has focused on trying to solve the problems of applying this approach in a variable bandwidth environment.

One issue to be considered in the variable network environment is how closely routing and QoS mechanisms should interact. One approach is to have them tightly coupled, in other words, support QoS routing. In principle, given a sufficiently rapid QoS-aware routing algorithm, whenever link conditions or network topology change, the routing algorithm would immediately find new routes through the network with sufficient resources to allow QoS commitments made by the network to be maintained. QoS routing is a challenging problem even in a static network; it is especially challenging in a dynamic one. Work on QoS routing in mobile *ad-hoc* networks has been documented in several papers (e.g., [3]–[5]).

Another option is to completely decouple QoS and routing. This approach is less difficult than QoS routing and is taken with RSVP [6] in traditional networks, which use “soft state” to reserve and release resources on a given path. Traditional routing protocols are used to route both RSVP and data traffic; when a change in routing occurs, RSVP traffic will follow the new path and reserve resources on the new path, while resources on the old path will “time out.” The INSIGNIA inband signaling system for supporting QoS in mobile *ad-hoc* networks [7] also assumes a decoupling of routing and QoS. INSIGNIA relies on soft state for dealing with the problem of changing resources. INSIGNIA also includes mechanisms for signaling QoS requirements along the new route using information incorporated into the data packet headers and application adaptability based on hierarchical flows.

Several efforts have focused on the more constrained problem of topology changes when only the end systems move. In this case, the problem becomes one of maintaining QoS when an end

system node is handed off from one access node to another. One approach that tackles the problem of maintaining QoS during handoffs assumes that, at least for some users, mobility will be predictable [8]. An implementation of this approach, called mobile RSVP (MRSVP), is based on modified versions of mobile IP and RSVP [9]. The MRSVP approach offers promise in dealing with the problem of handoff of mobile hosts from the access point in one cell to the next, but it still does not provide a mechanism for dealing with changes in link bandwidth within a cell.

A second approach that handles both handoff and variable link quality introduces the concept of adaptively re-adjusting the QoS within pre-negotiated bounds [10], [11]. We believe that this concept (treating reservations as ranges and allowing the network the flexibility to adjust QoS within this range) is crucial to dealing with variable link bandwidth, and we have adopted it as the basis for our work. The protocol developed as part of this approach also includes an algorithm for determining the bandwidth to allocate to each flow within the requested range.

Our approach, which we call dynamic QoS, is reservation-based and includes the notion of QoS ranges, although it uses a slightly different interpretation of a QoS range than that presented in previous papers [10], [11]. Our interpretation is that the network provides service at a signaled QoS allocation point within the range requested in the QoS reservation request. Service is guaranteed¹ at the allocated point, but the network may change this allocation point at any time. As long as the application is capable of adapting its transmission characteristics to stay within its allocated level, it receives QoS support from the network. Our protocol also includes an algorithm for determining the bandwidth to allocate to each flow within the requested range. This protocol and the bandwidth allocation algorithm are described in Section IV.

One reason that we find the notion of QoS ranges especially attractive is that it facilitates the decoupling of routing and QoS maintenance. If a change in network topology causes a new route to be computed, or if throughput changes on one of the links within a route, having a range rather than a single value increases the likelihood that QoS can be maintained at some point within the range. If resources decrease, the current allocation within the range can be decreased, rather than having to fail and tear down the reservation; and if resources increase, the current allocation can be increased accordingly.

It is important to note that our approach relies on “soft state” to reestablish QoS along the new route when a change occurs. When this happens, we rely on the concept of adaptive QoS to deal with the fact that a different level of resources may be available along the new route. However, the reliance on soft-state mechanisms means that when a route changes, there will be a period during which the traffic receives only best effort service. We believe that there is a significant class of applications that can tolerate transient periods of degraded service, yet benefit from dynamic QoS support.

The dynamic QoS approach requires that applications be able to specify their QoS needs as ranges rather than scalar values

¹We use the term “guarantee” loosely here; the exact meaning of the term “guarantee” depends on the service model.

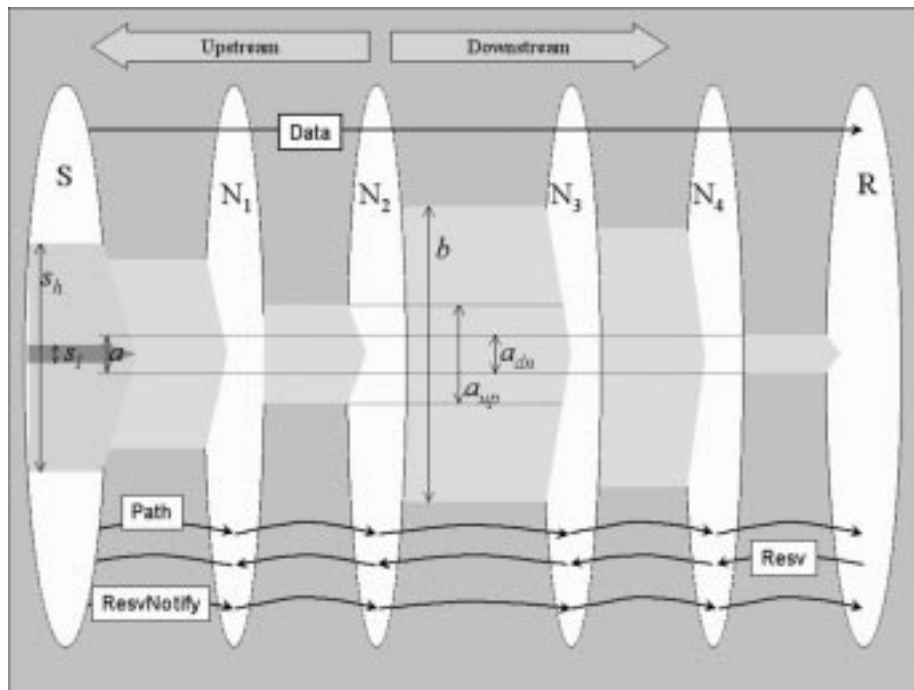


Fig. 1. Overview of dRSVP.

and be able to adapt to a changing QoS allocation. Later in the paper, we will discuss how we added adaptive behavior to existing streaming audio and video applications.

Our approach makes the following assumptions about the link layer: it deals with errors, it can provide information on resulting effective link bandwidth (similar to that described in [12]), and it can provide QoS support in a shared media network environment.

IV. DYNAMIC RSVP PROTOCOL (DRSVP)

To demonstrate the feasibility of the dynamic QoS approach discussed above, we defined a distributed network protocol that we call dRSVP. As the name suggests, this protocol is an extension of RSVP [6]. We implemented dRSVP by modifying and extending Information Sciences Institute's (ISI's) implementation of RSVP [13]. This implementation includes the controlled load service model [14], and the key managed resource at each router is interface bandwidth. 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], [15].)

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

- 1) We added an additional flow specification (FLOWSPEC) in Resv messages and an additional traffic specification (SENDER_TSPEC) in Path messages, so that they describe ranges of traffic flows.
- 2) We added a "measurement specification" (MSPEC) to the Resv messages, which is used to allow nodes to learn about "downstream" resource bottlenecks.
- 3) We created a new reservation notification (ResvNotify) message, which carries a "sender measurement specifica-

tion" (SENDER_MSPEC) that is used to allow nodes to learn about "upstream" resource bottlenecks.

- 4) We changed the admission control processing to deal with bandwidth ranges.
- 5) We added 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.
- 6) We extended SCRAPI, the simplified RSVP API [16], to deal with bandwidth ranges.

These extensions and modifications to RSVP comprise the dynamic RSVP (dRSVP) protocol, which is described below.

A. dRSVP Protocol Description

Fig. 1 illustrates a simple network in which node S sends data to node R through intermediate nodes N₁, N₂, N₃, and N₄. 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 travel the reverse of the route followed by the Path messages (assuming bi-directional links). Finally, ResvNotify messages flow through the network from S to R.

We will examine the operation of the protocol in detail, using the figure to illustrate how the protocol would operate at node N₂ for this simple example. Each node receives Path and ResvNotify messages from upstream nodes, and Resv messages

from downstream nodes. (The “upstream” and “downstream” directions are defined relative to the flow of *data* from S to R, not relative to the flow of protocol messages.) In the simple example shown in the figure, there is only a single flow, and each node has only one upstream and one downstream interface for this flow. In general, however, there will be multiple flows, and each flow may be multicast, so each flow can have multiple upstream interfaces and multiple downstream interfaces. In this case, a node could possibly receive different values of s_l and s_h in Resv messages from downstream receivers. Each node aggregates and stores the received values. We use $s_h(f)$ and $s_l(f)$ to denote the aggregated value of s_h and s_l for flow f at a given node. This aggregation is performed by setting $s_h(f)$ to the maximum² value of s_h received for flow f on any interface at the node, and setting $s_l(f)$ to the minimum s_l value received.

When a node receives a Resv message on interface i for flow f , requesting a resource reservation in the range (s_l, s_h) , it must determine how much bandwidth within this range it can allocate for the flow on that interface. It does this by executing a bandwidth allocation algorithm that divides up the available bandwidth on interface i , denoted $b(i)$, among all the flows that are utilizing this interface. This bandwidth allocation algorithm is a key part of the dRSVP protocol operation. The following discussion describes how we compute the bandwidth allocation for flow f on interface i , denoted $a(f, i)$.

First, if we have enough bandwidth on interface i to provide every flow on that interface with the maximum desired bandwidth, the bandwidth allocation algorithm will be simple because there is plenty of bandwidth to spare. Let $F(i)$ denote the set of all flows that have been admitted on downstream interface i . Then, the amount of bandwidth H needed to satisfy the maximum requested for all flows is given by

$$H = \sum_{g \in F(i)} s_h(g). \quad (1)$$

If $H \leq b(i)$, we simply allocate for f on interface i the maximum requested bandwidth

$$a(f, i) = s_h(f). \quad (2)$$

This is the case at node N_2 in Fig. 1. There is only one flow present, and there is sufficient bandwidth available on the downstream interface from N_2 to satisfy the maximum requested for the flow.

If there is not have enough bandwidth for this, i.e., $H > b(i)$, we look to see if there are flows that do not need the maximum requested bandwidth allocated, because they cannot utilize it due to bottlenecks elsewhere in the network. Resv messages received from downstream nodes contain a parameter, denoted m_r , that provides an indication of downstream bottlenecks. Similarly, ResvNotify messages received from upstream

nodes contain a parameter m_s that provides an indication of upstream bottlenecks. We use the notation $a_{dn}(g, j)$ to denote the value of m_r that we have received for flow g on downstream interface j . Similarly, $a_{up}(g, j)$ denotes the value of m_s that we have received for flow g on upstream interface j . (Later, we will describe how we report these values upstream and downstream.) Note that senders such as S in Fig. 1 do not have any upstream nodes. In this case, we simply set a_{up} to s_h , the maximum rate requested for the flow. Similarly, at receivers we set a_{dn} to s_h .

A multicast flow may have multiple upstream and downstream interfaces, so we need to aggregate the a_{up} and a_{dn} values for these different interfaces to determine the bottlenecks that may affect this flow elsewhere in the network. If we denote the set of downstream and upstream interfaces for flow g as $D(g)$ and $U(g)$, respectively, then we perform this aggregation as follows:

$$a_{dn}(g) = \min_{j \in D(g)} [a_{dn}(g, j)] \quad (3)$$

$$a_{up}(g) = \max_{j \in U(g)} [a_{up}(g, j)]. \quad (4)$$

We use the minimum when aggregating downstream values because we assume that the sending application will back off to the rate that can be reliably delivered to *all* receivers. As a result, there will be no need to reserve more bandwidth than could be delivered to the most constrained receiver. We use the maximum when aggregating upstream values because we want to ensure that we have reserved enough capacity to allow us to deliver the traffic received from the most aggressive transmitter.

Fig. 1 illustrates the values of a_{up} and a_{dn} at node N_2 for the single flow in the example.

Using the aggregated downstream and upstream bottleneck parameters, we can now obtain a single estimate of the bottlenecks that affect a flow elsewhere in the network. We refer to this estimate as the “external allocation” and it is computed simply as

$$a_{ext}(g) = \min [a_{up}(g), a_{dn}(g)]. \quad (5)$$

If we have enough bandwidth on interface i to provide every flow on that interface with at least as much as its external allocation, then we are not creating a bottleneck for any flow. The amount of bandwidth needed to satisfy the external allocation for all flows is given by

$$A_{ext} = \sum_{f \in F(i)} a_{ext}(f). \quad (6)$$

If $A_{ext} \leq b(i)$, then we can give each flow at least its external allocation. To avoid creating a bottleneck, we only need to reserve at least $a_{ext}(f)$ for flow f . However, even though we do not need to reserve more than the external allocation, we do want to advertise the fact that we *could* reserve more if we needed to. This is crucial to fast convergence of the distributed algorithm when bottlenecks in the network are removed. We need to report the fact that, at this node, the maximum reservation that we *could* give to each flow is its external allocation *plus* a share of the “excess” bandwidth available at this node. We assume that the excess bandwidth will be divided up among all the flows in

²For simplicity of exposition, we treat s_l and s_h as scalars. In fact, they are traffic specifications that contain not only average bandwidth, but also other parameters such as bucket size and peak rate; thus, the terms “maximum” and “minimum” are ambiguous. Throughout this paper, we use the terms “minimum” and “maximum” as a simplification for a traffic specification merging process as described in [14].

a proportionate manner, so the allocation at this node for flow f is given by

$$a(f, i) = a_{\text{ext}}(f) + \beta [s_h(f) - a_{\text{ext}}(f)]. \quad (7)$$

Here, β is a factor that determines how much each flow can be given of its requested range above the external allocation, computed as follows:

$$\beta = \frac{b(i) - A_{\text{ext}}}{H - A_{\text{ext}}}. \quad (8)$$

This formula divides up all the available bandwidth among all the flows on the interface. This can easily be seen by simply summing the bandwidth allocation over all flows

$$\begin{aligned} & \sum_{f \in F(i)} a(f, i) \\ &= \sum_{f \in F(i)} a_{\text{ext}}(f) + \sum_{f \in F(i)} \beta [s_h(f) - a_{\text{ext}}(f)] \\ &= \sum_{f \in F(i)} a_{\text{ext}}(f) + \beta \left[\sum_{f \in F(i)} s_h(f) - \sum_{f \in F(i)} a_{\text{ext}}(f) \right] \\ &= A_{\text{ext}} + \beta [H - A_{\text{ext}}] \\ &= A_{\text{ext}} + \frac{b(i) - A_{\text{ext}}}{H - A_{\text{ext}}} [H - A_{\text{ext}}] = b(i). \end{aligned} \quad (9)$$

Finally, if $A_{\text{ext}} > b(i)$, we indeed have a bottleneck on interface i . In this case, we compute L , the bandwidth needed in order to provide each flow with the minimum required bandwidth

$$L = \sum_{f \in F(i)} s_l(f). \quad (10)$$

If $L \leq b(i)$, then we give each flow the minimum of its range and divvy up any remainder proportionately

$$a(f, i) = s_l(f) + \beta [a_{\text{ext}}(f) - s_l(f)] \quad (11)$$

where

$$\beta = \frac{b(i) - L}{A_{\text{ext}} - L}.$$

As before, this formula divides up all the available bandwidth among all the flows on the interface.

On the other hand, if $L > b(i)$, there is insufficient capacity to maintain even the minimum. In this case, we reject flow f . If it is a new flow, it is considered to have failed admission control. If it is an existing flow, link bandwidth has decreased to the point that we cannot maintain the minimum requested bandwidth for all flows and some existing flow must be torn down. Our implementation simply tears down the first flow for which this condition is detected. A more sophisticated implementation would select a flow to tear down based on some policy, e.g., tearing down flows that are under utilizing their reservation or are somehow considered to be of lower priority than other flows.

Having computed the allocation for flow f , we know what level of resources to reserve. We also must determine what values to report as m_r and m_s for this flow in Resv and ResvNotify messages that we send upstream and downstream for this flow. To do this, we first take the minimum of the allocations on all of the downstream interfaces for the flow

$$a(f) = \min_{i \in D(f)} [a(f, i)]. \quad (12)$$

Then, the value of m_r we will report upstream is the minimum of the allocation we have made and the allocation made by other nodes downstream of us

$$m_r = \min [a(f), a_{dn}(f)]. \quad (13)$$

Similarly, the value of m_s we will report downstream is the minimum of the allocation we have made and the allocation made by other nodes upstream of us

$$m_s = \min [a(f), a_{up}(f)]. \quad (14)$$

In the example of Fig. 1, node N_2 is not a bottleneck, so it simply forward the value a_{dn} in its reports upstream and a_{up} in its reports downstream. Observe that N_2 knows of the existence and magnitude of the bottlenecks that are present in the network upstream and downstream from it. Fig. 2 shows the situation that would occur if a new flow were added that also traversed the link from N_2 to N_3 . Node N_2 must now decide how to divide the bandwidth of this interface between flows f and g . Since flow f is limited by external bottlenecks, N_2 knows that all the bandwidth above level $a_{dn}(f)$, as shown, can be allocated to flow g without affecting the end-to-end reservation for f . If the resources requested by g were high enough or if additional flows were added across this link or if the bandwidth of the link were to decrease, N_2 might find that the bandwidth it could allocate to f was less than the value of a_{dn} . Node N_2 would then become the new bottleneck for flow f , which would be reported in Resv and ResvNotify messages forwarded by N_2 , affecting the end-to-end reservation level for the flow.

B. dRSVP Response to Link Variability

The dRSVP protocol assumes that the link layer detects and responds to link variations and reports the results to the network layer. For example, for a wireless link, the link layer is assumed to adjust encoding or medium access control behavior as needed to maintain low error rates and to report to the network layer the resulting effective transmission data rate available. When this rate changes, dRSVP responds by adjusting the bandwidth allocated to each flow that traverses the link. Care must be taken that transient changes in link characteristics do not result in excessive protocol overhead and unnecessary application adaptations. This can be avoided in two ways. First, the link layer itself should include algorithms (e.g., damping and hysteresis) to avoid reporting transient changes to the network layer. Second, the dRSVP algorithm only recomputes bandwidth allocations for a given flow when a reservation refresh occurs. The reservation refresh time interval needs to be set short enough to ensure that the protocol is sufficiently responsive to network conditions, but not so short as to create excessive protocol overhead. The nominal RSVP refresh interval is 30 s; in our testbed, we used a refresh rate of 10 s to increase responsiveness to link variability.

C. dRSVP Application Programming Interface

For the application to signal its requirements and to adapt to dynamic network conditions, the API between RSVP and the application needed to be modified. We extended a current RSVP API to include this capability, creating an API called the dRSVP application programming interface (dSCRAPI). This API is based on the SCRAPI interface provided with ISI's RSVP

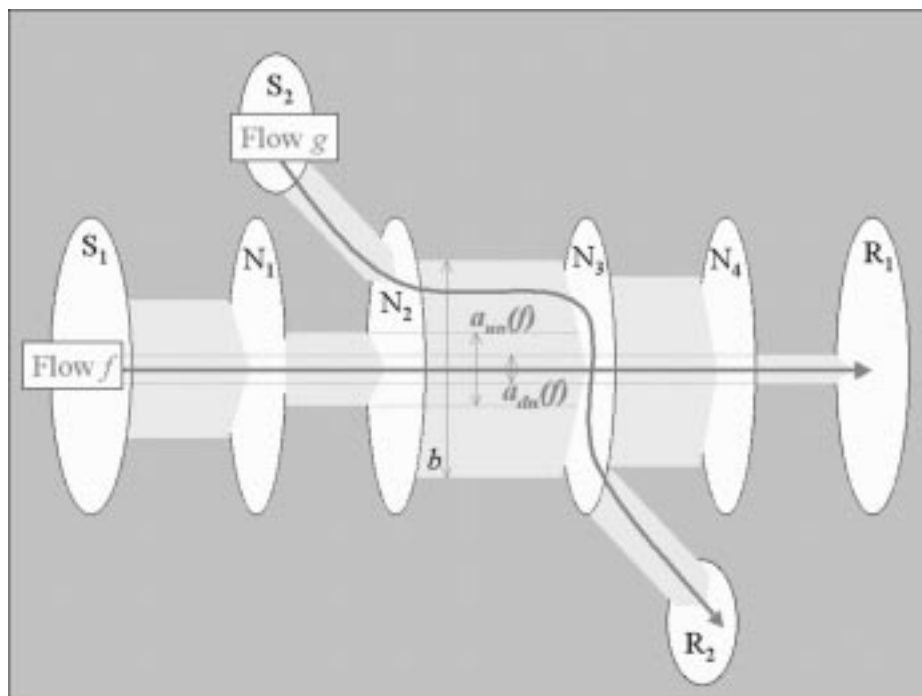


Fig. 2. Example with a second flow.

implementation [16]. Our API allows an application to specify the range of bandwidths within which it is capable of operating and to request QoS support for operation within this range. A “callback” mechanism is provided to allow the application to learn the status of a reservation request and to learn the current allocated bandwidth within the requested range. The application can then adapt its transmission rate to the allocated level and receive QoS support for its traffic.

An interesting feature of dSCRAPI is that we were able to make it “backward compatible” with SCRAPI. We did this by retaining all of the existing SCRAPI function calls unmodified. Within dSCRAPI, the old functions are mapped into the new dSCRAPI functions by translating scalar QoS values into ranges with the high end and the low end both set to the same value. For example, a SCRAPI function for requesting bandwidth X is mapped into a dSCRAPI function for requesting bandwidth in the range (X, X) , yielding exactly the desired behavior from dRSVP. Using this technique, we were able to link existing unmodified RSVP-aware applications with dSCRAPI instead of SCRAPI and then run these applications in our dRSVP testbed. (A more complicated issue is compatibility between dRSVP routers and standard RSVP routers. This issue is discussed in Section VII.)

V. IMPLEMENTATION STATUS AND DISCUSSION

A. Testbed

In order to test and evaluate our implementation of the adaptive QoS protocol described in Section IV, we have created a testbed in which we can vary resources available in a network of routers. These routers can be configured in a variety of complex network topologies. The routers are Intel-based PCs running FreeBSD with the alternate queuing (ALTQ) package installed [17], [18].

Our testbed includes mobile nodes connected via 802.11 wireless LAN interfaces, but the capability we relied on most to test and refine the protocol was to emulate, on wired ethernet links, the effects of dynamic link characteristics. The dynamic link emulation capability includes a centralized testbed controller application, which 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, shown in Fig. 3, sends commands to a bandwidth manager daemon resident on each node in the testbed. The commands are sent over a 100-Mb/s ethernet control LAN to which every node in the testbed is attached (not shown in figure). The daemon implements the link speed change command by interacting with the ALTQ package. The link speed change is effected by modifying the queue service parameters used in class-based queuing (CBQ) [19]. 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 underserved, i.e., they actually receive less bandwidth than specified. This is due to limitations of the ALTQ implementation, which uses a preconfigured “average packet size” in the transmission scheduling algorithm rather than the actual size of each packet transmitted. This makes the implementation more efficient, but causes it to underserve queues with many small packets and overserve queues with large packets.

B. Adaptive Applications

In order to obtain insight into the value of dynamic QoS for a realistic application, we selected the user datagram protocol (UDP)/real time transport protocol (RTP)-based streaming

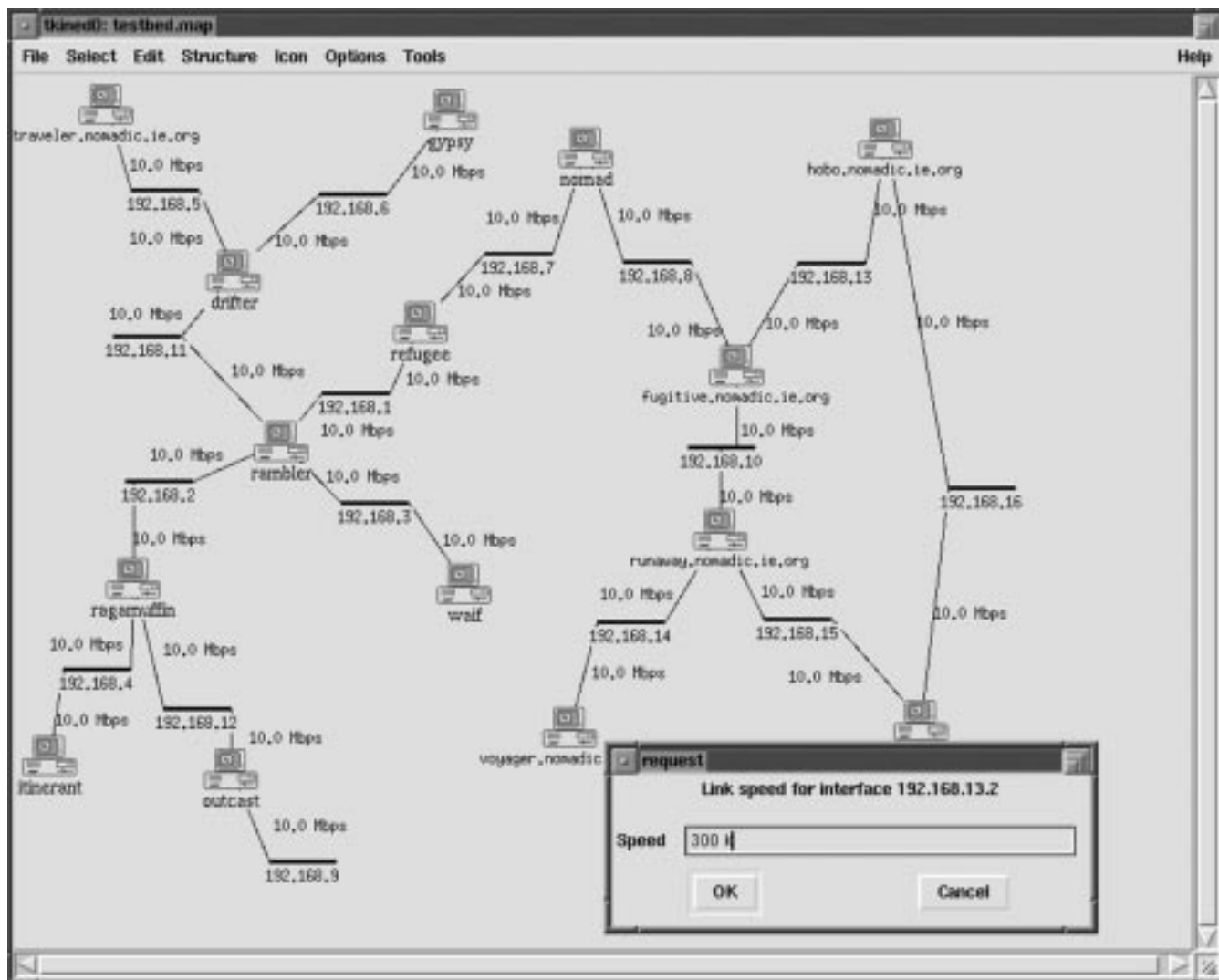


Fig. 3. Testbed controller application.

video and audio applications vic [20] and vat [21], which were created at Lawrence Berkeley National Laboratory and modified to be RSVP-aware by ISI. We further modified these applications to make them adaptive and to work with dRSVP.

Fig. 4 shows a screen shot of our modified version of vic. When the "Transmit" button is selected by the user, vic operates in a conventional (nonadaptive) mode. In this mode, the application transmits at a data rate selected by the user using the "rate control" slider, and this is the rate for which a reservation will be issued when the user selects the "reserve" button at the receiver. On the other hand, when the "Adaptive Transmit" button is selected and the "reserve" button is clicked at the receiver, our version of vic operates in an adaptive mode. In this mode, the application requests a reservation for a bandwidth range determined by the extremes of the "rate control" slider (10 kb/s to 1 Mb/s in the example shown in Fig. 4). Using information obtained from the dSCRAPI API, this application will automatically adjust the frame rate and image quality to stay within the bandwidth allocation provided by dRSVP, and the rate control slider moves to show the current allocation. The algorithm for selecting frame rate and image quality is quite simple; as

available bandwidth falls, the application reduces the frame rate until a preconfigured minimum frame rate is reached. Below the threshold minimum frame rate, the application begins to reduce image resolution. Other adjustments, e.g., reducing image size or selecting different compression algorithms, could also be implemented. However, the simple algorithm we implemented to adjust frame rate and resolution is quite effective in maintaining good video quality as perceived by the user throughout a wide range of available bandwidth.

With the audio tool vat, adaptation occurs by selecting an audio encoding to stay within the allocated bandwidth. The range of bandwidths in the reservation requests issued by vat, when operating in adaptive mode, is simply the range from the most compact encoding (about 8 kb/s) to the least compact encoding (about 78 kb/s).

For both vic and vat, the programming effort required to make the applications adaptive was quite modest. This was partly due to the fact that these applications were already written to be capable of operating at different speeds. There may be a large class of streaming applications that would lend themselves to this type of adaptive behavior.



Fig. 4. Adaptive video application.

In addition to adaptive versions of *vic* and *vat*, we also created adaptive versions of the test tools “*mgen*” and “*drec*,” which allow us to create unicast and multicast streams of UDP traffic and to install RSVP reservations for these flows. Our adaptive versions of *mgen* and *drec*, which we named “*dmgen*” and “*ddrec*,” function like *mgen* and *drec* except that *ddrec* is capable of issuing reservation requests containing a bandwidth range and *dmgen* is capable of adapting its transmission rate according to the available QoS signaled by the *dRSVP* daemon. These tools proved valuable for creating controlled tests and gathering quantitative performance data, as described below.

C. Implementation Over 802.11 Wireless LAN

While the link emulation technique described above provides an effective way to test and evaluate the protocol, an implementation over actual wireless hardware was considered important to fully demonstrate the feasibility of implementing the protocol. For this purpose, we chose Lucent WaveLan (now called “Orinoco”) wireless LAN cards. In order to utilize

these cards we had to modify the device driver software to allow us to obtain feedback from the card on the throughput it was achieving. This proved to be extremely difficult because of the lack of detailed technical information available from the manufacturer, who considers the interface to the card to be proprietary. However, by reverse engineering the existing FreeBSD device drivers (which others had created by reverse engineering the Linux device drivers) we were able to make modifications that allowed us to obtain in real time the raw transmission rate (1, 2, 5.5, or 11 Mb/s) being used by the card for each packet. The card selects the transmission rate through a proprietary mechanism that maintains a low packet loss rate. In accordance with the 802.11 protocol, the card also uses an acknowledgment mechanism to provide reliable packet transmission for all unicast traffic. We were unable to obtain any information from the card on the number of retransmissions that were occurring; therefore, we were only able to form a crude approximation of the effective throughput based on the raw transmission rate. Nevertheless, even with only this limited information available, we were able to write a link-monitoring tool that provided inputs to the *dRSVP* bandwidth allocation algorithm. The throughput estimates provided by this tool are crude, but nevertheless allowed us to demonstrate the protocol working correctly over the wireless link and to demonstrate visible qualitative improvements in application performance.

Fig. 5 shows a snapshot of the output of the link-monitoring tool. This snapshot was taken as we walked down the hallway carrying a laptop equipped with a wireless LAN card while transmitting video over the wireless LAN interface. The top chart in the figure shows the estimate of available bandwidth decreasing as the range to the peer node increases. The bottom chart shows counts of the packets transmitted at each of the raw rates that the card is capable of using.

We should note that our implementation did not include providing any form of “subnet bandwidth manager” to prevent other nodes from using bandwidth that had been reserved for flows outbound from a given node. We were able to work within this limitation by simply restricting our demonstration and test scenarios to include only a pair of wireless LAN cards, with the bulk of the traffic flowing in one direction. A complete dynamic QoS implementation over 802.11 would include a bandwidth manager built on top of the 802.11 point coordination function (PCF), which provides contention-free access to the medium. Due to the lack of technical information available from the card’s manufacturer, we were unable to utilize the features of the PCF.³ Thus, we were limited to the contention-based distributed coordination function (DCF). Nevertheless, our implementation showed the feasibility of implementing the *dRSVP* protocol and adaptive applications over actual wireless network hardware.

VI. PROTOCOL PERFORMANCE

A. Qualitative Evaluations

Tests and demonstrations performed in our testbed comparing standard and *dRSVP* have illustrated the benefits of the

³We were unable to ascertain from Lucent whether the WaveLan/Orinoco cards implement the PCF.

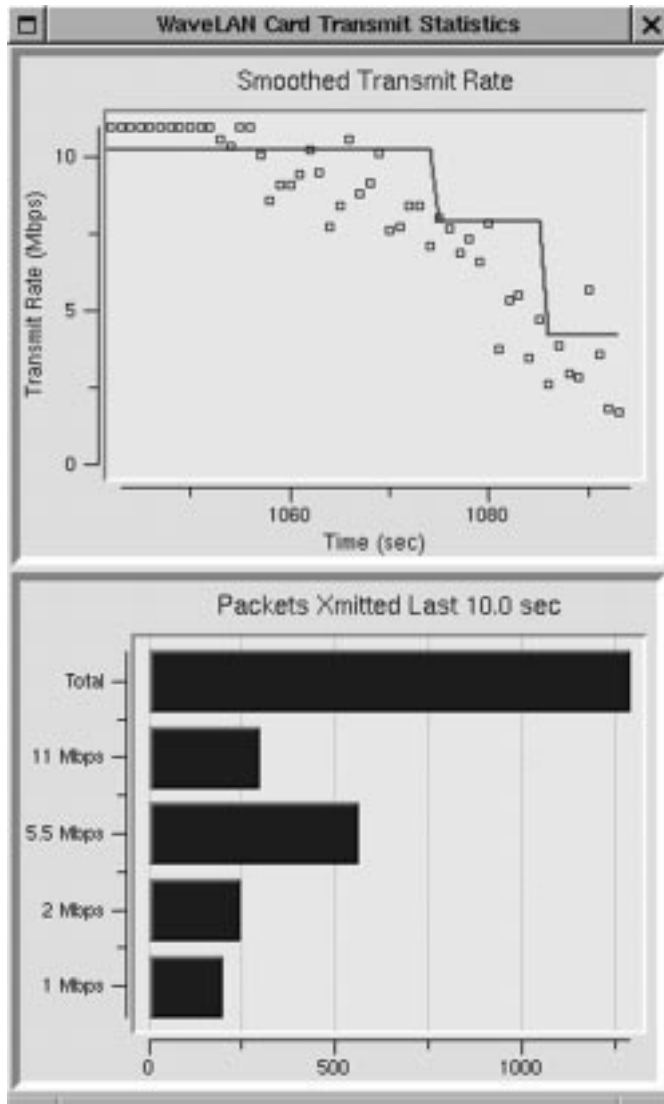


Fig. 5. Wireless LAN monitoring tool.

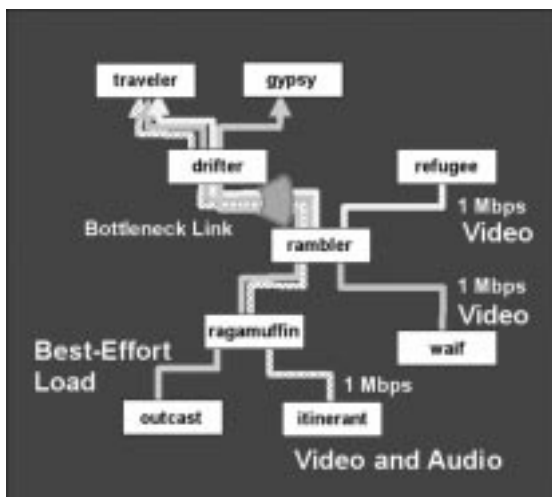


Fig. 6. Sample demonstration configuration.

adaptive QoS approach in a varying bandwidth environment. Fig. 6 shows one of the configurations we used in our testbed for these demonstrations. In this configuration, we generated

three different multicast video sessions, originating at the machines named itinerant, waif, and refugee, and subscribed to by traveler, as shown. Simultaneously, we generated a flow of best effort traffic from outcast to gypsy. We used our testbed controller to vary link speeds to create and remove bottlenecks on the links traversed by the flows, as indicated by the funnel shape icon on the link from rambler to drifter. We can also inject audio flows into the network. With a configuration such as this, we demonstrated how dRSVP divides available bandwidth among several applications and responds to varying link speeds. We also showed how the applications adapt to variable bandwidth allocations, continuing to receive QoS support even under degraded conditions. Demonstrations such as this provided convincing qualitative evidence of the value of the dynamic QoS approach.

We also conducted qualitative evaluations using the wireless LAN implementation. In these evaluations, we transmitted video (generated from a camera focused at a moving fan) over the wireless LAN to a laptop. We then walked down the hall carrying the laptop and subjectively observed the video performance. Standard RSVP was simply useless in this environment, as the link bandwidth varied so widely that there was no meaningful fixed value to use to configure RSVP with the link bandwidth. As we walked down the hall (or when someone turned on the microwave oven in the nearby staff kitchen) the link bandwidth would decrease, resulting in a “breakup” of the video image. With dRSVP, on the other hand, the video image would not break up, but would smoothly and gracefully degrade to lower frame rate and then lower resolution as we walked away with the laptop. The demonstration dramatically illustrated the inherent value of an adaptive application combined with a link throughput signaling mechanism.

B. Quantitative Performance Results

In addition to qualitative demonstrations of the advantages of the dynamic QoS approach, we also performed experiments to gather quantitative measures of protocol performance. In particular, we were interested in gaining insight into the following questions.

- 1) How does the performance of an application using dRSVP compare with using standard RSVP or no reservation protocol in a variable bandwidth environment under essentially identical conditions?
- 2) What impact does dRSVP itself have on the network compared to RSVP?

To help evaluate these questions, we defined and executed a set of incremental experiments in our testbed. The Expect [22] toolkit was used to automate the execution of experiments and each experiment was executed at least ten times. Adaptive and nonadaptive versions of mgen and drec were used to generate traffic flows. We defined five factors of particular interest initially:

- 1) configuration;
- 2) type of flow (i.e., best-effort, standard RSVP, dRSVP);
- 3) number of flows (1, 2, ...);
- 4) link fluctuation scenario (none, slow, fast);
- 5) distribution of flow (unicast, multicast).

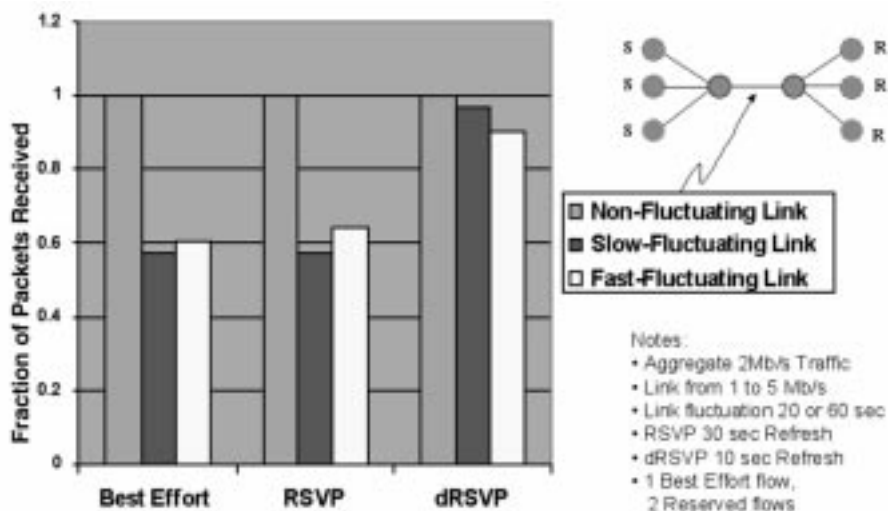


Fig. 7. Reliability.

The metrics we were interested in fell into two classes: user performance metrics and network performance metrics. We examined several metrics in these categories, but focused on just two. The user performance metric we focused on was the fraction of transmitted data successfully received. The network performance metric we focused on was overhead bytes per flow per second.

Another metric we examined was CPU utilization of designated nodes, but we did not observe any significant change in CPU utilization from standard RSVP to dRSVP.

Fig. 7 shows reliability (fraction of packets received) results with a test configuration as shown. In the “Best Effort” test, a total of 2 Mb/s of best effort traffic was sent from the nodes designated “S” (senders) to the nodes designated “R” (receivers). In the RSVP test, a best-effort flow and two controlled load flows were injected into the network, again with a total traffic load of 2 Mb/s. The dRSVP test was similar to the RSVP test, with one best effort flow and two controlled load flows generating up to 2 Mb/s of traffic, but in this case, the applications generating the controlled load flows were adaptive, so they could “back off” in response to dRSVP signaling. For each of these tests, the “bottleneck” link bandwidth was either held constant at 5 Mb/s (“nonfluctuating”) or varied between 1 and 5 Mb/s every 20 s (fast fluctuating) or every 60 s (slow fluctuating), creating the nine test cases shown.⁴ The results of these tests confirm our expectations. First, with a nonfluctuating link, we see no packet loss, since the link capacity is more than adequate to carry all flows. With fluctuating links, both the best effort and RSVP cases show significant packet loss; only about 60% of application traffic actually gets through. Note that in this case, RSVP offers almost no advantage over best effort. For both best effort and RSVP cases, the fraction of packets received was slightly greater with fast fluctuating links. We surmise that this is due to buffering in the router. For the dRSVP case we see a significant improvement over standard RSVP for both slow and fast fluctuating links. In this case, the fraction of packets received was

⁴These are actually just a subset of a much larger set of test cases selected for purposes of exposition.

slightly lower with fast fluctuating links than with slow fluctuations. We attribute this to the fact that it takes dRSVP some time to respond to link speed changes, resulting in some loss with each downward fluctuation.

These results show that dRSVP, when used by an adaptive application, can result in significant improvement in the effective reliability provided by the network. This improvement, of course, does not come for free. Fig. 8 shows the link bandwidth overhead consumed by the signaling protocol for the same test cases as described above. For best effort there is, of course, no overhead. For dRSVP there is significantly more overhead than for standard RSVP. This increased overhead is attributed to multiple sources. First, the dRSVP messages are slightly larger than standard RSVP messages, due to the need to carry bandwidth ranges rather than scalar values. Second, in order to be responsive to link fluctuations, the refresh interval used with dRSVP was one third of that used in RSVP (10 s as opposed to 30 s). Last, every link fluctuation results in additional signaling as the dRSVP protocol communicates new bandwidth allocations throughout the network. This last factor also explains why the overhead for dRSVP increases with the rate of link fluctuation, as shown in the test results.

The dramatic increase in overhead of dRSVP compared with standard RSVP could be seen as a major disadvantage of our approach. However, it is important to also look at the absolute scale. Even in the worst case, the total overhead was only around 70 B/s per flow, or 560 b/s per flow. This overhead is not dependent on the data rate, but is dependent on the refresh rate, which in turn would be set according to the expected rate of link fluctuation and desired responsiveness of the protocol. For low-speed links (e.g., 2.4 kb/s) with high fluctuation rates the overhead may be a problem, but for higher link speeds (e.g., 56 kb/s or higher) and slower fluctuation rates the overhead may be considered negligible.

Fig. 9 illustrates some dRSVP overhead results for different numbers and types of flows. Case 1 consists of a single unicast adaptive flow, while Case 2 consists of two unicast flows: one adaptive and one best effort. Both Cases 1 and 2 expe-

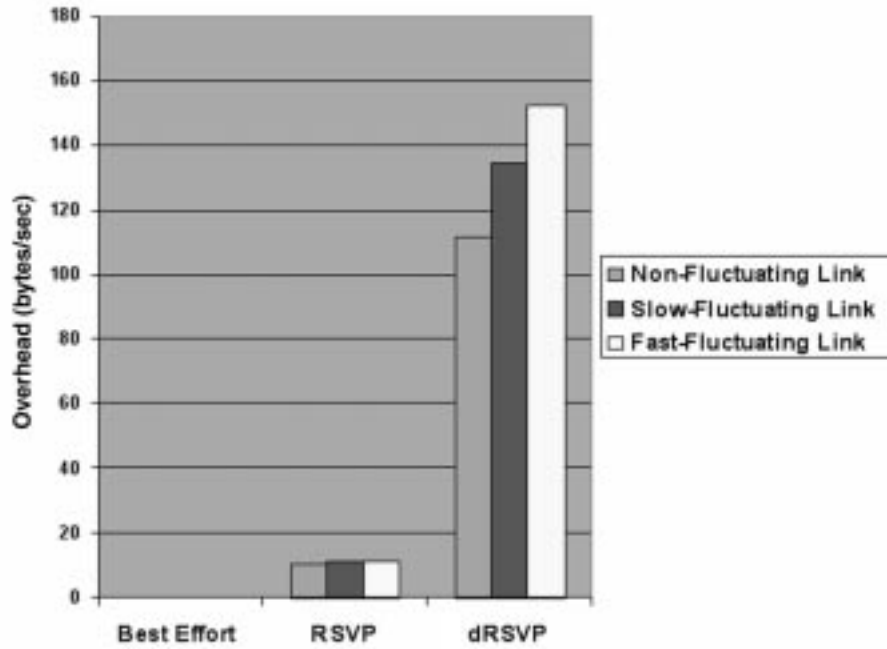


Fig. 8. Overhead.

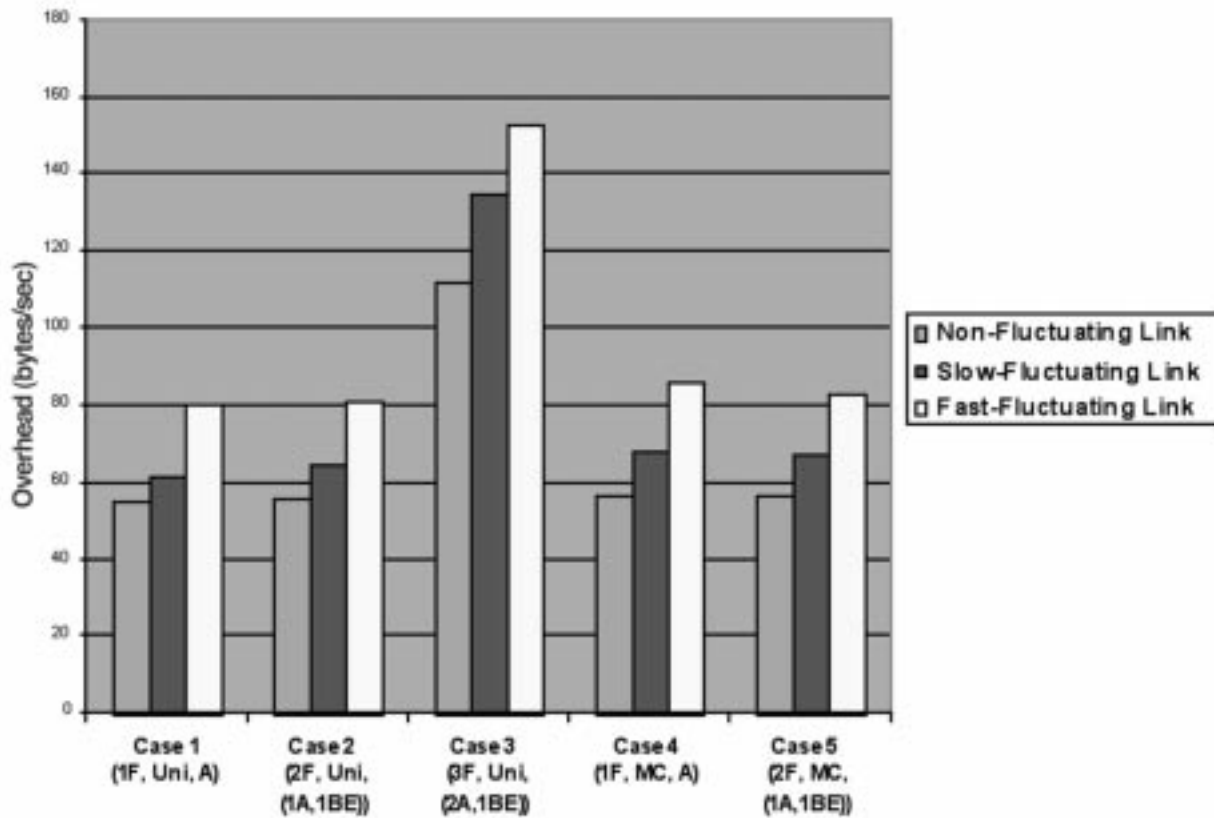


Fig. 9. Adaptive flow overhead cases.

rience essentially the same amount of overhead, even though the amount of “reserved” data sent is twice as much for Case 1 than for Case 2. Similar results are obtained for Cases 4 and 5, both of which include a single multicast adaptive flow (Case 5 also includes one best effort unicast flow). However,

when the number of adaptive flows is increased from one to two (from Case 2 to 3), so does the overhead, with the Case 3 overhead approximately twice the overhead for Case 2. This implies a linear scaling when the number of adaptive flows increases. Fig. 10 illustrates this point further for an

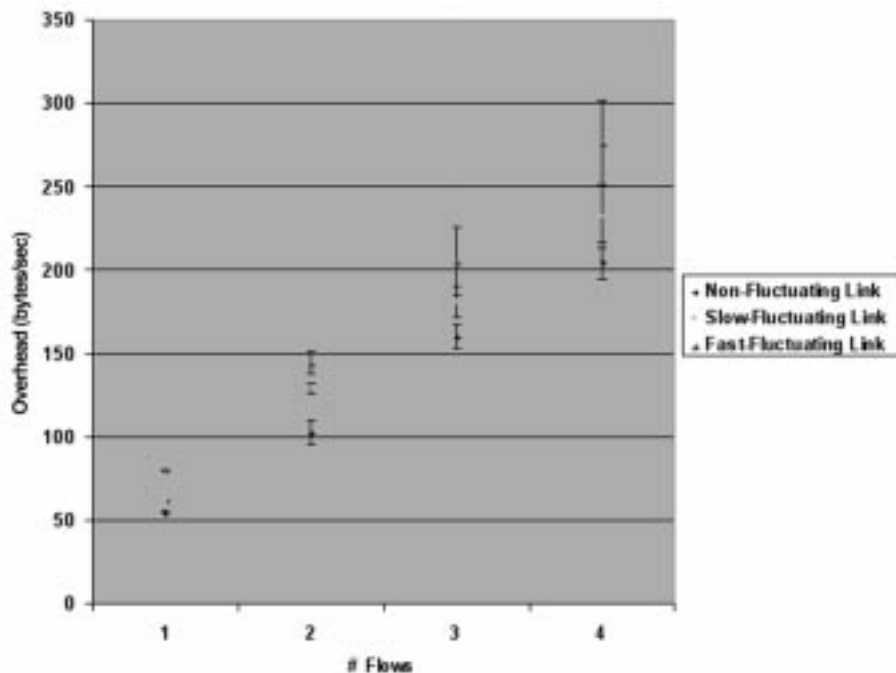


Fig. 10. Increasing adaptive unicast flows.

increasing number of adaptive unicast flows (with 90% confidence intervals calculated).

VII. DEPLOYMENT AND INTEROPERABILITY WITH STANDARD RSVP

We envision two scenarios for deployment of dRSVP. In the first, dRSVP is used in isolated stand-alone networks built out of custom routers that all support dRSVP. This scenario might be valuable for special purpose networks where QoS is critical and link bandwidth is highly variable, e.g., military networks. The disadvantage of the scenario is that it assumes special-purpose routers with custom development to provide the dRSVP, support and these are less desirable from a cost and support standpoint than standard products. In the second scenario, dRSVP is used in networks in which certain routers support dRSVP and others are standard routers supporting standard RSVP. The routers that support dRSVP would be those attached to critical bottleneck links or to variable (e.g., wireless) links. The network would support both standard end systems and dRSVP-aware end systems, with the latter hosting both adaptive and nonadaptive applications. This scenario is attractive because it would allow gradual introduction of dRSVP and allow utilization of standard routers within networks that also use custom dRSVP routers. This assumes that dRSVP can interoperate with standard RSVP in a meaningful way, which we believe is possible. We should be clear that our testbed implementation of dRSVP is not interoperable with standard RSVP, simply because we took the fastest path to implementing the basic dRSVP functionality. However, the code could be modified to interoperate with standard RSVP. In the following paragraphs, we discuss how interoperability between dRSVP and standard RSVP could be achieved.

To achieve interoperability between dRSVP and standard RSVP, at least two issues need to be addressed. First, the extra

information defined by dRSVP (e.g., bandwidth ranges rather than scalars) needs to be carried transparently through portions of the network that contain standard RSVP routers. Second, we need to define how dRSVP routers should interpret standard RSVP messages.

Allowing dRSVP information to pass through standard RSVP routers is possible due to the foresight of the RSVP protocol designers [6]. An RSVP message consists of a common header, followed by a body consisting of a variable number of variable-length typed “objects.” Each object begins with an object header that contains a “Class-Num” field and a “C-Type” field that identifies the object. If an RSVP implementation receives an unknown object with the two high-order bits of the Class-Num field set, it is required to “ignore the object but forward it, unexamined and unmodified, in all messages resulting from this message.” We can take advantage of this rule to allow the additional information needed by dRSVP to be carried transparently through standard RSVP routers. In standard RSVP, the bandwidth request is carried in a FLOWSPEC object that is carried in the Resv message. Our approach would be to place the high end of a dynamic QoS bandwidth range in the standard FLOWSPEC object and to place the low end of the bandwidth range in a new FLOWSPEC_LOW object. Similarly, in Path messages, we would place the high end of the bandwidth range in the standard SENDER_TSPEC object and the low end of the bandwidth range in a new SENDER_TSPEC_LOW objects. The new objects would be given a new Class-Num with high order bits indicating “ignore but forward if unknown.” In addition to bandwidth requests being ranges rather than scalars, dRSVP has additional requirements for passing information related to upstream and downstream bottlenecks, the parameters m_s and m_r . To carry this information, we use a new MSPEC object in the Resv message to carry the downstream bottleneck parameter m_r and a new SENDER_MSPEC object in the Path

message⁵ to carry the upstream bottleneck parameter m_s . For these new objects, we would also assign a new Class-Num with “ignore but forward if unknown” semantics. Since these objects will be ignored by standard RSVP, it is critical that dRSVP be running on routers attached to potential bottleneck links in order to have the characteristics of these links addressed.

The second issue that needs to be addressed in this scenario is that a dRSVP router may receive standard RSVP messages that do not contain the extra objects needed for the dRSVP protocol. In particular, instead of a bandwidth range, the dRSVP router will only receive a scalar bandwidth value. The dRSVP router can easily handle this case by simply assuming a trivial range with high and low values set to the single received value. Other parameters needed by dRSVP can be handled in a similar manner.

Using the approach described above, the additional information needed by dRSVP would be forwarded through portions of the network containing standard RSVP routers. If an adaptive application generated a dRSVP request along a path containing standard RSVP routers, these routers would see only standard FLOWSPEC and SENDER_TSPEC objects containing the high end of the requested bandwidth range. On the other hand, dRSVP routers along the path would see the complete set of information required for the operation of the dRSVP protocol. At the links connected to the standard RSVP routers, the flows could only be admitted providing sufficient bandwidth was available to satisfy the upper end of the requested range, but the dRSVP routers could operate the full dRSVP protocol as described in this paper, and would only need to allocate some level of bandwidth within the requested range. In a scenario where standard RSVP routers are used on high capacity links with plenty of bandwidth and dRSVP routers are used on the critical or variable links, this approach would work well.

VIII. CONCLUDING REMARKS

We set out to discover how reservation-based QoS mechanisms could be applied under the dynamic conditions created by wireless networks. We believe that the key concept in solving this problem is to adopt the notion of reservations as ranges rather than as scalar values. This approach provides the necessary flexibility that allows network elements and end systems to adapt to varying network conditions.

With our testbed, adaptive applications, and dRSVP implementation, we have been 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. Our protocol implementation is built as an extension to RSVP, which was chosen as a convenient and logical base for work on resource reservation-based QoS mechanisms. We have discussed how dRSVP routers could be made interoperable with standard RSVP routers and we have

⁵In our current implementation, m_s is carried in a new ResvNotify message. However, the “ignore but forward if unknown” mechanism applies only to unknown objects within known message types, not to new message types. Therefore, to use this mechanism for interoperability purposes, we would have to change our implementation to carry m_s within the existing Path message rather than within a new ResvNotify message.

described a possible scenario for gradual deployment of dRSVP. While the time may not be right for wide deployment of dRSVP, we hope that our work will be used in specialized applications and will lead to further research to create protocol standards that could be applied more generally in future architectures in which QoS capabilities are needed to support multimedia applications across networks that include wireless links with dynamic characteristics.

Many interesting possibilities remain open for investigation. One possible area 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. Another possible area is integrating an adaptive QoS approach with a (separate) QoS routing solution. Still another is 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.

ACKNOWLEDGMENT

The authors would like to thank the contributions of R. Moose and J. Andresen on this project.

REFERENCES

- [1] R. Bagrodia, W. Chu, L. Kleinrock, and C. Popek, “Vision, issues, and architecture for nomadic computing [and communications],” *IEEE Pers. Commun.*, vol. 2, pp. 14–27, Dec. 1995.
- [2] M. Stemm and R. Katz, “Vertical handoffs in wireless overlay networks,” *ACM Mobile Networks Applicat.*, vol. 3, no. 4, pp. 335–350, 1998.
- [3] S. Chen and K. Nahrstedt, “Distributed quality-of-service routing in *ad-hoc* networks,” *IEEE J. Select. Areas Commun.*, vol. 17, pp. 1488–1505, Aug. 1999.
- [4] C. Lin and J. Liu, “QoS routing in *ad-hoc* wireless networks,” *IEEE J. Select. Areas Commun.*, vol. 17, pp. 1426–1438, Aug. 1999.
- [5] R. Sivakumar, P. Sinha, and V. Bharghavan, “CEDAR: A core-extraction distributed *ad-hoc* routing algorithm,” *IEEE J. Select. Areas Commun.*, vol. 17, pp. 1454–1465, Aug. 1999.
- [6] S. Jamin, “Resource ReSerVation Protocol (RSVP)—Version 1 Functional Specification,” Network Working Group, Internet Engineering Task Force, RFC 2205, Sept. 1997.
- [7] S. Lee and A. Campbell, “INSIGNIA: In-band signaling support for QoS in mobile *ad-hoc* networks,” in *Proc. 5th Int. Workshop Mobile Multimedia Communications*, Berlin, Germany, Oct. 1998.
- [8] A. Talukdar, B. Badrinath, Rutgers Univ., and A. Acharya, C&C Research Labs, “On accommodating mobile hosts in an integrated services packet network,” in *Proc. INFOCOM*, Apr. 1997.
- [9] A. Talukdar and B. Badrinath, “MRSVP: A reservation protocol for an integrated services packet network with mobile hosts,” Dept. Comput. Sci., Rutgers Univ., New Brunswick, NJ, Tech. Rep. DCS-TR-337, July 1997.
- [10] S. Lu and V. Bharghavan, “Adaptive resource management algorithms for indoor mobile computing environments,” in *Proc. ACM SIGCOMM*, Aug. 1996, pp. 231–242.
- [11] S. Lu, K. Lee, and V. Bharghavan, “Adaptive service in mobile computing environments,” in *Building QoS Into Distributed Systems*, A. Campbell and K. Nahrstedt, Eds. London, U.K.: Chapman & Hall, 1997.
- [12] D. Beyer, T. Frivold, J. Hight, and M. Lewis. (1998, Sept.) API framework for Internet radios. [Online]. Available: ftp://ftp.rooftop.com/pub/apis/api_framework.pdf.
- [13] . RSVP implementation. Inf. Sci. Inst., Univ. Southern California, Los Angeles, CA. [Online]. Available: <http://www.isi.edu/div7/rsvp/rsvp.html>.
- [14] J. Wroclawski, “Specification of the controlled-load network element service,” Internet Engineering Task Force, RFC 2211, Sept. 1997.

- [15] R. Braden and L. Zhang, "Resource ReSerVation Protocol (RSVP)—Version 1 Message Processing Rules," Internet Engineering Task Force, RFC 2209, Sept. 1997.
- [16] B. Lindell and ISI. (1998, August) SCRAPI—A simple "bare bones" API for RSVP, work in progress (draft-lindell-rsvp-scrapi-00.txt). [Online]. Available: <http://www.isi.edu/rsvp/DOCUMENTS/draft-lindell-rsvp-scrapi-02.txt>.
- [17] K. Cho, "A framework for alternate queueing: Toward traffic management by PC-UNIX based routers," in *Proc. USENIX 1998 Annu. Technical Conf.*, New Orleans, LA, June 1998, [Online]. Available: www.csl.sony.co.jp/~kjc/kjc/papers.html.
- [18] —, "Managing traffic with ALTQ," in *Proc. USENIX 1999 Annu. Technical Conf.*, Monterey, CA, June 1999, [Online]. Available: www.csl.sony.co.jp/~kjc/kjc/papers.html.
- [19] S. Floyd and V. Jacobson, "Link-sharing and resource management modules for packet networks," *IEEE/ACM Trans. Networking*, vol. 3, pp. 365–386, Aug. 1995.
- [20] S. McCanne and V. Jacobson, "vic: A flexible framework for packet video," in *Proc. 3rd ACM Int. Conf. Multimedia*, Nov. 1995, pp. 511–522.
- [21] V. Jacobson and S. McCanne, *vat—X11-Based Audio Teleconferencing Tool*, Lawrence Berkeley Nat. Lab., Berkeley, CA, 1994.
- [22] D. Libes, *Exploring Expect.* Cambridge, MA: O'Reilly, 1995.



Mohammad Mirhakkak (S'79–M'81) received the B.S. and M.S. degrees in electrical engineering from Tehran University, Tehran, Iran, in 1970 and 1971, respectively, and the D.Sc. degree in computer science from The George Washington University, Washington, DC, in 1981.

He has been with the MITRE Corporation, McLean, VA, since January 1990. His current research interests include development of routing protocols for mobile networks, simulation and evaluation of network protocols, quality-of-service protocols, and network security.



Nancy Schult (A'97) received the B.S. degree in mathematics from Meredith College, Raleigh, NC, and the M.C.S. and Ph.D. degrees in computer science from the University of Virginia, Charlottesville.

She is currently with the MITRE Corporation, McLean, VA. Her current research interests include network protocols for wireless and wired communications, quality-of-service issues in *ad-hoc* networks, and simulation and performance evaluation of networks.



Duncan Thomson received the B.S. degree in mathematics from the University of Washington, Seattle, in 1980 and the M.S. degree from Northeastern University, Boston, MA, in 1986.

He began his career working on Kalman filter-based cooperative tracking systems at General Dynamics Electronics in 1981. Since 1983, he has worked for the MITRE Corporation, McLean, VA, in the areas of air defense systems, air traffic control and aeronautical networks, and military tactical networking and communications. He has been active

in building prototype systems and experimental testbeds, beginning in 1984 with a prototype gateway for special-purpose air defense communications systems. From 1991 to 1993, he led the Aeronautical Telecommunication Network (ATN) Project and helped create a testbed and MITRE prototype of an ATN router. His current research interests include areas such as the interaction between network and link-layer quality of service, mobile routing with directional antennae, and applications of RSVP. He is currently involved in a project creating IP interfaces to various satellite communications services.