Dynamic Quality of Service and Adaptive Applications for a Variable Bandwidth Environment

Mohammad Mirhakkak, Nancy Schult, Duncan Thomson
The MITRE Corporation

Abstract

This paper 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 QoS approach centers on the notion of providing QoS support at some point within a range requested by applications. The applications must be capable of adapting to the level of QoS provided by the network, which may vary during the course of a connection. The paper describes a new protocol called dynamic RSVP (dRSVP) that we have implemented to demonstrate and evaluate the dynamic QoS concept. The protocol and a new application programming interface (API) for this dynamic QoS are described. Finally, we briefly discuss our experience with adaptive streaming video and audio applications that work with the new protocol in a testbed network.

1. Introduction

A number of challenges exist in providing support for nomadic computing. This support has been summarized as [1]:

"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 potentially moving nodes. This network environment can include variable link characteristics and connectivity changes, resulting in variations in available bandwidth.

In this paper, we first discuss variable bandwidth in nomadic networks and some issues involved in providing Quality of Service (QoS) support in this environment. We then define an approach for providing QoS support in a dynamic network environment that is adaptive within a range requested by an application. With this QoS model, applications must also 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, dynamic RSVP (dRSVP) that supports this paradigm, which is the primary focus of this paper. Implications for adaptive applications are discussed. Finally, we briefly discuss our experience with adaptive streaming video and audio applications that are UDP-based and work with the new protocol in a testbed network.

2. 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, resulting in changes in transmission quality. If the lower layers do not detect or respond to changes in transmission quality, 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 to deal mainly with congestion loss and network layer queueing 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 (ARQ), 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, not only can the bandwidth of individual links change, but end-to-end bandwidth can change even when links remain stable, when topology changes result in a new route through the network which traverses links with different available resources.

3. 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 variation 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 how our approach is 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. This approach is attractive for applications that need a per-flow, end-to-end QoS solution (as opposed to the approach based on providing individual packets with different per-hop behaviors at a given node, depending on type of service markings on individual packets). However, a resource reservation-based approach is problematic in a variable bandwidth environment. If available resources vary after admission control has been performed, then the network may find that it is unable to meet commitments for flows that have been successfully admitted.

One issue to be considered in the dynamic network environment deals with 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 changes, 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],[4],[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 uses "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, while the old resources will "time-out". The INSIGNIA in-band 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, in which 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 quality of service 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 this concept as the basis for our work. Their protocol 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 is slightly different 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 QoS reservation request. Service is "guaranteed"[1] 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 the following section.

One reason that we find the notion of QoS ranges especially attractive is that it facilitates the decoupling of routing and

[1] We use the term "guarantee" loosely here; the exact meaning of the term "guarantee" depends on the service model.

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.

A Dynamic QoS approach implies the need for applications to be able to specify their QoS needs as ranges rather than scalar values, and 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.

Note that we make the following assumptions about link-layer capabilities: 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.

4. The Dynamic RSVP (dRSVP) Protocol

To demonstrate the feasibility of the Dynamic QoS approach discussed above, we defined in 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 dRSVP by modifying and extending ISI's implementation of RSVP [13]. This implementation includes the Controlled Load service model [14], and we assume that the resource of interest is 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:

a) We added 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) We added a "measurement specification" (mspec) to the Resv messages, which is used to allow nodes to learn about "downstream" resource bottlenecks.

c) We created a new reservation notification (ResvNotify) message, which carries a "sender measurement specification" (smspec) that is used to allow nodes to learn about "upstream" resource bottlenecks.

d) We changed the admission control processing to deal with bandwidth ranges.

e) 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.

f) We extended SCRAPI, the simplified RSVP API [16], to deal with bandwidth ranges.

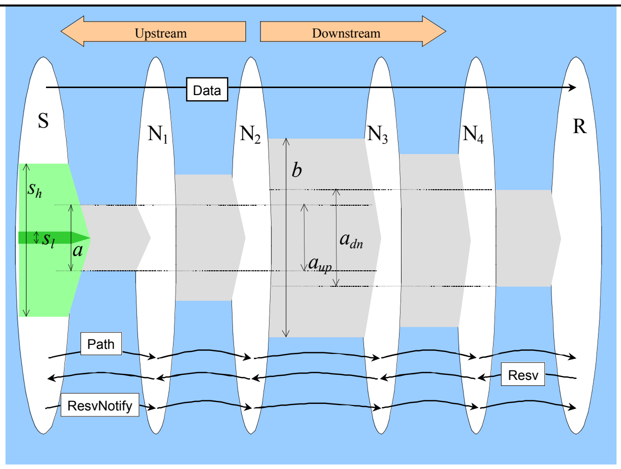

Figure 1. Overview of dRSVP

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

4.1. dRSVP Protocol Description

Figure 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 is the reverse of the route followed by the Path messages (assuming bi-directional links). Finally, Resv-Notify 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 N_2 for this simple example. Each node receives Path and Resv-Notify 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 (how values are aggregated is discussed later). We use $s_h(f)$ to denote the maximum[2] value of s_h that has been received for flow f, and $s_l(f)$ to denote the minimum value of s_l that has been received for flow f at a given node.

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 that 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:

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

[2] For 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 document we use the terms "minimum" and "maximum" as a simplification for a traffic specification merging process as described in RFC 2211 [14].

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

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

This is the case at node N_2 in Figure 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, Resv-Notify 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 Figure 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 define this aggregation as follows:

$$a_{dn}(g) = \min_{j \in D(g)} \left[a_{dn}(g,j) \right] ,$$
$$a_{up}(g) = \max_{j \in U(g)} \left[a_{up}(g,j) \right] .$$

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.

Figure 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:

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

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:

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

If $A_{ext} \le b(i)$, then we can give each flow at least its external allocation. This will provide flow *f* with enough bandwidth to avoid running up against bottlenecks elsewhere in the network. To avoid creating a bottleneck, we only need to reserve at least a_{ext} 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 a proportionate manner, so the allocation at this node for flow *f* is given by:

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

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:

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

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

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

Finally, if $A_{ext} > b(i)$, 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:

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

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

$$a(f,i) = s_l(f) + \beta\left[a_{ext}(f) - s_l(f)\right], \text{ where}$$
$$\beta = \frac{b(i) - L}{A_{ext} - L}$$

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, for example tearing down flows which are underutilizing their reservation or are somehow considered as being 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 Resv-Notify 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:

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

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:

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

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:

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

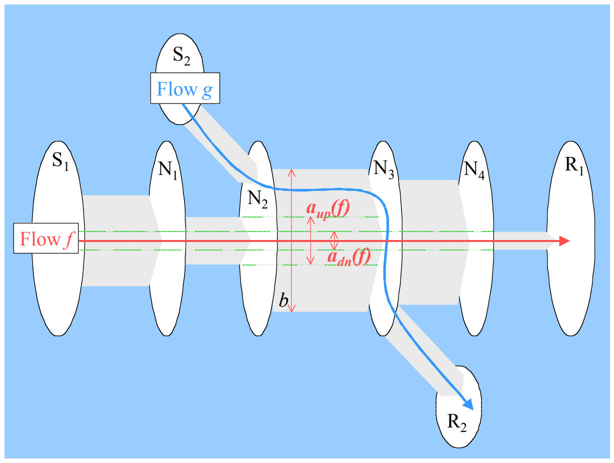

Figure 2. Example with a Second Flow

In the example of Figure 1, node N_2 is not a bottleneck, so it simply forwards 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. Figure 2 shows the situation that would occur if a new flow was 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 bottlnecks, 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 Resv-Notify messages forwarded by N_2, affecting the end to end reservation level for the flow.

4.2 dRSVP Application Programming Interface

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

5. Implementation Status and Discussion

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 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 queueing (ALTQ) package installed [17][18].

Our testbed does not yet include mobile nodes, but it does include the ability to emulate the effects of dynamic link characteristics. To accomplish this we created 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 sends

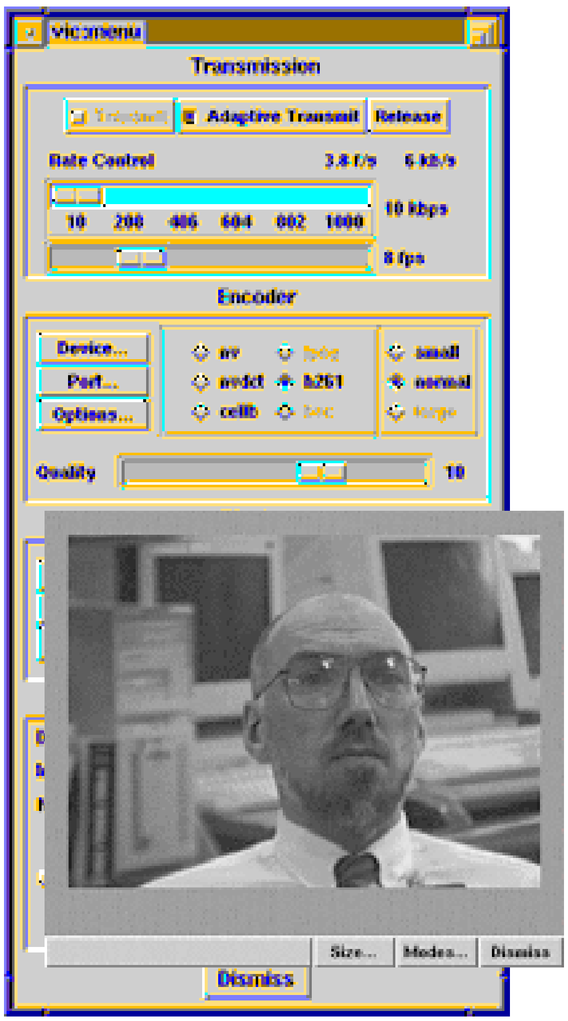

Figure 3. Screenshot of "vic" Video Application

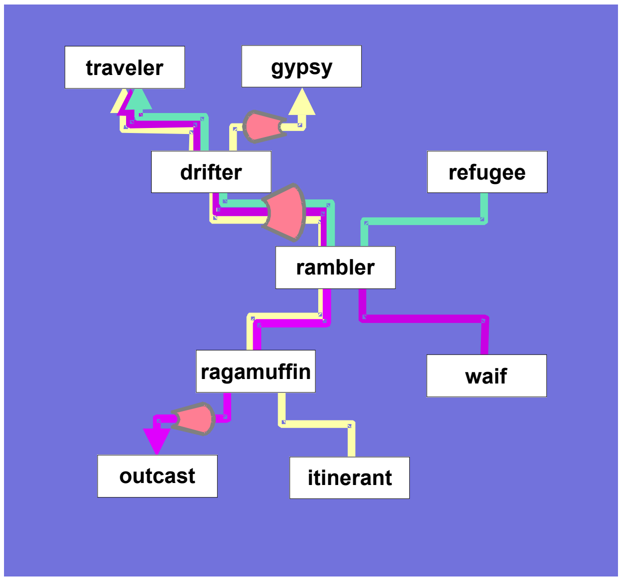

Figure 4. Sample Demonstration Configuration

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 [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 under-served; that is, they actually receive less bandwidth than specified.

In order to obtain insight into the value of dynamic QoS for a realistic application, we selected the UDP/RTP-based streaming video and audio applications vic and vat, which were created at LBL and modified to be RSVP-aware by ISI. We modified these applications to make them adaptive and work with dRSVP.

Figure 3 shows a screen shot of our modified version of vic. When the "Transmit" button is selected by the user, 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, the application requests a reservation for a bandwidth range determined by the extremes of the "rate control" slider (10 Kbps to 1 Mbps in the example shown in Figure 3). Using information obtained from the dSCRAPI API, this application will automatically adjust the frame rate to stay within the bandwidth allocation provided by dRSVP, and the rate control slider moves to show the current allocation. 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 8Kbps) to the least compact encoding (about 78Kbps).

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. Also, we should note that these applications use Real-time Transport Protocol (RTP) and UDP as their underlying transport mechanism. We are currently implementing an adaptive web server application, which will give us experience in how TCP-based applications can operate with a dynamic QoS paradigm.

Tests and demonstrations performed in our testbed comparing standard and dynamic RSVP have illustrated the benefits of the adaptive QoS approach in a varying bandwidth environment. Figure 4 shows one of the configurations we have used in our testbed for these demonstrations. In this configuration we generate three different multicast video sessions, originating at the machines named itinerant, waif, and refugee, and subscribed to by outcast, traveler, and gypsy, as shown. We use 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 icons on the links. We can also inject best effort traffic into the network, as well as audio flows. With this configuration we can demonstrate how dRSVP divides available bandwidth among several applications and responds to varying link speeds. We also show how the applications adapt to variable bandwidth allocations, continuing to receive QoS support even under degraded conditions. Demonstrations such as this provide convincing qualitative evidence of the value of the dynamic QoS approach, and a quantitative evaluation is currently underway.

6. Concluding Remarks

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. 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 in order to analyze scalability. We also would like to add wireless hardware into our testbed, and experiment with node movement and a dynamic topology.

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.

References

[1] 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.

[2] M. Stemm, R. Katz; *Vertical Handoffs in Wireless Overlay Networks*; ACM Mobile Networking (MONET), Special Issue on Mobile Networking in the Internet; Winter 1998.

[3] 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.

[4] C. Lin, J. Liu, *QoS Routing in Ad Hoc Wireless Networks,* IEEE Journal on Selected Areas in Communications, Vol 17, No. 8, August 1999.

[5] 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.

[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] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] D. Beyer, T. Frivold, J. Hight, M. Lewis, API Framework for Internet Radios, September 1998. ftp://ftp.rooftop.com/pub/apis/api_framework.pdf .

[13] ISI, The RSVP Implementation developed by the University of Southern California (USC) Information Sciences Institute (ISI). http://www.isi.edu/div7/rsvp/rsvp.html

[14] J. Wroclawski, *Specification of the Controlled-Load Network Element Service*, Internet Engineering Task Force Request For Comments Number 2211, September 1997.

[15] R. Braden, L. Zhang; *Resource ReSerVation Protocol (RSVP) -- Version 1 Message Processing Rules*, IETF RFC 2209, September 1997.

[16] 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

[17] 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

[18] 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

[19] 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.