Two Issues in Reservation Establishment

Scott Shenker and Lee Breslau
Palo Alto Research Center
Xerox Corporation
Palo Alto, California 94304-1314
{shenker,breslau}@parc.xerox.com

Abstract

This paper addresses two issues related to resource reservation establishment in packet switched networks offering real-time services. The first issue arises out of the natural tension between the local nature of reservations (i.e., they control the service provided on a particular link) and the end-to-end nature of application service requirements. How do reservation establishment protocols enable applications to receive their desired end-to-end service? We review the current one-pass and two-pass approaches, and then propose a new hybrid approach called one-pass-with-advertising. The second issue in reservation establishment we consider arises from the inevitable heterogeneity in network router capabilities. Some routers and subnets in the Internet will support real-time services and others, such as ethernets, will not. How can a reservation establishment mechanism enable applications to achieve the end-to-end service they desire in the face of this heterogeneity? We propose an approach involving replacement services and advertising to build end-to-end service out of heterogeneous per-link service offerings.

1 Introduction

There are many proposed designs that make Internet-style packet-switched networks capable of supporting real-time applications (see [3, 6, 11, 12, 14, 18, 19, 21, 22, 23, 25] and references therein for a few representative examples). These proposals typically involve three main changes to the Internet architecture. The first change is to extend the Internet service model beyond its current single class of *best-effort* service to include delay-bounded and other packet delivery services.[1] These services are typically defined in terms of end-to-end quantities; for instance, the delay bounds in a delay-bounded service refer to the total end-to-end delays experienced by packets.

The second change is in the routers. In order to provide these new services, routers may need to implement packet scheduling algorithms that are more sophisticated than FIFO. In addition, each router must have an admission control module which can respond to requests for service. Admission control is needed if any of the services provide quantitative assurances about the actual level of service provided, such as a delay bound, rather than just a description of the level of effort, such as a priority level. Such an admission control module not only responds with a yes/no to the request, but also determines what service must be installed at the router. While services are typically defined on an end-to-end basis, all services are necessarily implemented on a per-link basis, and so end-to-end service requirements must be translated into the necessary service performed at that router. Thus, along with the service model comes a set of *router requirements* that dictates what each router along the path must do in order to meet the end-to-end service requirements.

The third change is that a reservation establishment protocol must be introduced to propagate the client's service request to the switches along the data flow path. The reservation protocol must take a service request issued by an application and then must visit each router along the path delivering an appropriate service request.[2]

In this paper, we discuss two issues that arise when designing such a reservation establishment protocol. The first issue involves the relationship between end-to-end services and their per-link implementations. Applications care only about the end-to-end quality of service their flows receive. How does the reservation establishment protocol, in conjunction with the admission control modules on the routers, enable applications to get the end-to-end service they desire? In particular, if an application desires a certain end-to-end delay bound, how do the appropriate per-link services get installed along the path to make sure this bound is met? This question is especially difficult in the context of multicast [7] flows, where there are many service requests (e.g.,

This research was supported in part by the Advanced Research Projects Agency, monitored by Fort Huachuca under contract DABT63-94-C-0073. The views expressed here do not reflect the position or policy of the U.S. government.

[1] It is important to note that not all proposals involve extending the service model. As discussed in [26], one possible approach is merely to implement Fair Queuing in all switches and then put the onus on applications to be sufficiently adaptable. We confine our attention in this paper to proposals that do include adding delay-bounded and other similar packet delivery services to the service model.

[2] Our language here is intentionally vague because some reservation protocols deliver the application's service request to each router unchanged, whereas other reservation protocols take the application service request and then do some computation on that request before delivering it to individual routers. We discuss this more fully in Section 3.

one from each receiver) to deal with simultaneously.

The second issue involves the set of services itself. When the service model is extended beyond its current single class of best-effort service, there will inevitably be heterogeneity in the set of services supported by different routers. Some routers will support the entire service model while others, such as ethernets, will continue to support only the traditional best-effort service. The reservation establishment protocol, again in conjunction with the admission control modules on the routers, must manage this heterogeneity carefully. We want applications to be minimally disturbed by this heterogeneity. In addition, we also want to maximize the extent to which partially deployed services are utilized, and to facilitate incremental deployment of new services.

These two issues will arise in the context of any architecture that supports real-time applications. The architecture must enable applications to secure the end-to-end services they desire, and it must also be able to cope with the inevitable presence of heterogeneity in the capability of network routers. These two problems are not adequately addressed by existing proposals. In this paper, we propose a general approach to these two problems, which provides applications with knowledge about the end-to-end service that will result from a particular service request and enables applications to build end-to-end service out of heterogeneous service offerings at different routers.

This paper has 5 sections. In the next section we describe the context in which we discuss these problems. In Section 3 we discuss the problem of mapping end-to-end requests into per-link services. We first argue that the existing approaches, which we categorize as one-pass and two-pass, do not provide the desired functionality. We then propose a new approach, which we call one-pass-with-advertising (OPWA). In Section 4 we discuss the problem of heterogeneity. Our approach here, which uses the notion of replacement services and advertising, builds end-to-end service out of heterogeneous service offerings. We conclude in Section 5 with a brief discussion of our findings.

2 Context

The problems we address in this paper are rather general, and we think they will arise in the context of any service model that includes services explicitly designed to support real-time applications using resource reservation. However, it is difficult to discuss these issues absent the context provided by actual mechanisms or proposals. Consequently, we have chosen to consider a rather specific context motivated by proposals resulting from previous research on service models and reservation establishment protocols.[3] At the end of sections 3 and 4 we will discuss what fundamental aspects of the service model and reservation establishment protocol are necessary for these issues to arise. We present the context in two parts: (1) the service model and the associated router requirements, and (2) the reservation establishment protocol. We then discuss the implications of reservations in a multicast environment, which raises the issues of sharing reservations among sources and merging reservations from different receivers.

[3] The two proposals we describe are also being considered by two IETF working groups: the Integrated Services Working Group (for the service model) and the RSVP Working Group (for the reservation establishment protocol).

2.1 Service Model and Router Requirements

The service model we use for context is based on that described in [6] and [25]. We assume that the network offers three basic packet delivery services related to real-time applications, which we describe below. This paper should not be read as an endorsement of, nor an argument for, this particular set of services, and we expect that the actual service model of the future Internet will be somewhat different than what is discussed here. We have chosen these services because they represent a concrete proposal and they are general enough to provide a useful illustration of the problems at hand.

Before describing these three services, we should note that each of the services requires the application to make a reservation, and for each reservation request routers must make admission control decisions. As part of the reservation request, the application must specify its traffic characteristics to the network; we call this traffic specification the TSpec. To specify traffic we use the token bucket filter [6, 21], which has two parameters: the token bucket rate r and the token bucket depth b. A source conforms to this filter if, and only if, the cumulative bits sent during any period of time (t, t'), denoted by $S(t, t')$, satisfies $S(t, t') \leq r(t' - t) + b$. Thus, r is an upper bound on the long term rate of the traffic and b bounds its burstiness.

The most traditional form of real-time service is what we call *guaranteed* service (see [6]), which provides an absolutely firm delay bound on every packet. The way this bound is provisioned is not completely traditional; rather than providing a delay bound at each switch and then adding up the delays, we take advantage of the celebrated result due to Parekh and Gallager [21, 22, 23] on the weighted-fair-queueing (WFQ) scheduling algorithm [8]. The guaranteed service delivered to an individual application is parametrized by a rate r_g (specified by the client) and error terms A and B (specified by the network). For a flow characterized by an (r, b) token bucket TSpec and a reserved rate r_g, with $r_g \geq r$, the maximal packet delay, following [21, 22, 23], is given by[4] $\frac{b}{r_g} + \frac{A}{r_g} + B$. To implement this end-to-end service, the switches are required to bound their error away from the weighted-fair-queueing (WFQ) fluid model ([8, 21, 22, 23]) in terms of A^α and B^α (where α is a label of the switch).[5] Then, the end-to-end error parameters are given by the sums: $A = \sum_\alpha^{alongpath} A^\alpha$, and $B = \sum_\alpha^{alongpath} B^\alpha$. This form of service allows the maximal end-to-end delay bound to be significantly less than the sum of the per-link delay bounds, in that the term $\frac{b}{r_g}$ only appears once. To summarize, the end-to-end definition of guaranteed service is that the delays will be bounded by $\frac{b}{r_g} + \frac{A}{r_g} + B$. The service required at each hop is that the service must model the fluid model to within an error tolerance defined by A^α and B^α.

A second type of real-time service, introduced in [6], is

[4] This is a slight generalization of their result, in that the error terms are more general.

[5] We do not explain more fully what the fluid model is, nor how error tolerances are described by the parameters A^α and B^α, because it would take us too far afield. We mention this service mainly because it has a nontrivial translation between per-link service and end-to-end service.

predictive service. Predictive service provides a fairly reliable bound on the maximal delay of each packet. This service is designed for those applications that are tolerant of occasional violations of the delay bound, and thus do not need the complete reliability of guaranteed service. The small reduction in reliability enables significantly higher network utilization, giving applications more efficient service. Routers offering predictive service provide three logical levels of service, each associated with its own delay bound.[6] The end-to-end delay bound is the sum of the per-link delay bounds.

An even less stringent form of real-time service is *controlled delay* service. This service is designed for applications that need some control on delay, but do not need specified delay bounds; the switch promises to provide some level of delay control, but the network makes no quantitative assurances about the end-to-end service characteristics. Thus, the network is only specifying a *level-of-effort* and not an actual delay bound. Routers supporting this service are required to use admission control to turn flows away when delays are too high, but there is no standard definition of what "too high" means. We again assume there are three logical levels of service provided by every router, which correspond to differing levels of delay control.

The service model we consider has three real-time services. Each service has a different relationship between the per-link service and the resulting end-to-end service; controlled delay has no quantitative end-to-end assurances, predictive service has an end-to-end delay bound which is the sum of the per-link delay bounds, and guaranteed service's end-to-end delay bound depends on the cumulative error terms A and B. These relationships between the end-to-end service and the per-link services are the only aspects of the service model that are essential to our present discussion.

In addition to these real-time services, we assume that the service model also includes three priority levels of the standard best-effort service.[7] This service, unlike those above, requires no admission control and the network makes no assurances about the resulting delays. Of course, the delays experienced in best-effort service depend on the ambient loading. Thus, while best-effort service provides no quantitative assurances, on a well-provisioned link it can provide very high quality service.

Because the service model provides the basic abstraction for networking service built into end host software, it must remain extremely stable. The service model is extensible, in that new services can be added, but existing services can not be easily removed. However, it is unrealistic to expect, in most circumstances, that these services will be uniformly and universally deployed. For instance, in the Internet context, the Internet Engineering Task Force (IETF) will likely *define* these services, but it will probably not mandate *deployment* of them (see [27]). That is, it will provide standards delineating what a router must do in order to claim to support specific services, such as guaranteed, predictive, and controlled delay. However, it will be up to individual router vendors and network service providers to decide which of these services is actually supported in any particular router. This leads to the issue of heterogeneity we discuss in Section 4.

[6] The choice of three levels is somewhat arbitrary, but there is a need for a standard number of levels.

[7] Reference [25] refers to this as ASAP service.

2.2 Reservation Protocol

We base our discussion of the resource reservation protocol on RSVP, which was introduced in [29] and is currently under discussion in the IETF (see [30] for a preliminary design specification). This protocol was designed to support both unicast and multicast applications, and thus serves as a useful guide to our discussion. For our purposes in this paper, there are three basic design principles that are especially relevant. First, the protocol is receiver-initiated, in that service requests are generated by receivers and then propagated towards senders. These service requests, carried in RESV messages, follow the reverse of the paths over which data packets travel. Second, the protocol is based on the notion of *soft-state* [5], which means that the state in the switches is periodically refreshed by the receivers. Rather than relying on the network to detect and respond to failures, RSVP puts the onus on receivers to resend their service requests periodically; if a failure has occurred a refreshed service request will re-establish the appropriate state. This is a very important design principle, since it requires that the reservation establishment messages be *idempotent*. That is, a service request must result in the same state being installed at a router whether the request is new or a retransmission. Third, the protocol itself is independent of the service model. While RSVP transports the client service requests to the routers, it need not understand the content of those requests.[8] This allows RSVP to remain unchanged while the service model is extended.

2.3 Implications of Multicast

The flows for which reservations are being made can be multicast flows, where each packet is transmitted down a distribution tree from a source to all members of the multicast group. There can be many sources of the flow, and these sources need not be members of the multicast group. This has two important implications for our discussion of reservation establishment. We do not go into the details here, but only give the following short description.

The presence of multiple sources raises the issue of *shared channels*. As proposed in [15] and [29], a given reservation can be shared by the traffic streams from several sources. Shared channels take advantage of application level semantics to reduce the aggregate resources reserved. For example, rather than reserving resources for each speaker in a multiparty conference, a single shared reservation, large enough to allow one or two simultaneous speakers, can be established. The Wildcard reservation style in RSVP shares a single reservation among all sources to a multicast group.

The presence of multiple recipients raises the issue of reservation merging. When two receivers request a reservation for the same flow, their requests must be combined into a single reservation at all links on their common path. Merging requires installing a service that is at least as *good* as either individual service request (i.e., an *upper bound*). Thus, there must be an ordering relationship between services so that a router can determine if it can establish such an upper bound on the service requests. Admission control must respect this ordering, in that if a flow would be ac-

[8] To be more precise, it must understand the *scope* of the requests, which is the list of the sources sharing this reservation, but it need not understand the quality of service requested for the flow.

cepted for a given level of service, it would also be accepted for all lower (relative to the ordering) service levels.

3 Satisfying End-to-End Service Requirements

There is an inherent tension in most approaches to resource reservations in networks. Reservations are local, in that they pertain only to the service provided on individual links. Yet, the purpose of reservations is to meet application requirements, and applications care only about the resulting end-to-end service. How does one design the reservation mechanism so that the set of per-link reservations results in the desired end-to-end service? This is the problem we discuss in this section.

We should note that this is only a problem for services where there is a nontrivial relationship between what is provided per-link and the resulting end-to-end service. By nontrivial here we mean that the application cannot compute, without additional knowledge, the per-link service that results in the desired end-to-end service: we will say such services have nontrivial composition properties. For a service completely characterized by a bandwidth reservation, the end-to-end service is merely the minimum of the bandwidths provided on each link, and so the reservation protocol would merely establish the same reserved bandwidth on each link. Hence, bandwidth has a trivial composition property. For services characterized by delay bounds, however, the relationship between the end-to-end delays and the per-link service is more complicated. With predictive service, for example, the end-to-end delay bound is the sum of the per-link delay bounds. Establishing the appropriate set of per-link bounds to meet a particular end-to-end delay target requires knowledge of the number of hops in the path and the latency along the path, and that is where the problem arises. This problem is not restricted to the provision of delay bounds. Even adaptive applications, which do not need delay bounds in order to process the incoming packets, face the issue of how to obtain the desired end-to-end delays.

The appendix contains a detailed discussion of what information applications need to know, both before and after making a reservation. For our purposes it is sufficient to assume that some real-time applications would prefer to know the end-to-end service that results from a given service request, and that others would like to know the resulting end-to-end service *before* selecting a service. An example of the former might be an audio playback application that values fidelity over reduced delay; such an application would like to know the maximum delays experienced by its packets so it can set its playback point. Applications with very tight delay constraints may want to know the resulting end-to-end delays before making their service requests in order to choose an appropriate service level.

We now turn to how to meet these needs. To make the presentation more precise, in this section we refer to the case of an application caring about the end-to-end delay bound (as opposed to other relevant parameters of the delivered service). In practice, reservation requests might come from some generalized resource manager on the host rather than the application itself; for convenience we will just refer to applications as the entities controlling the endpoint reservation process.

3.1 Current Approaches

There are many different reservation protocol designs (see, for example, [12, 13, 14, 18, 24, 28, 29]), and they differ in many important aspects. At a more abstract level, however, one can say that there are only two basic approaches to reservation establishment in the literature. The first is a one-pass mechanism, as represented by RSVP. The second is a two-pass mechanism, as represented by the original Tenet design [1, 16] and by ST [13, 28]. We discuss these approaches separately, and in each case ask to what extent they can meet the end-to-end service needs of applications. The sketchy descriptions we provide here merely capture the general properties of these approaches; we do not address any of the detailed aspects.

3.1.1 One-Pass Mechanisms

The current mechanism in RSVP delivers a reservation request sequentially to the admission control module in each switch along the data path, thereby making reservations along the way in a single pass. RSVP itself does not manipulate this reservation request; as far as RSVP is concerned, this reservation request is an opaque sequence of bits that RSVP deposits at each router. This reservation request could be modified by the admission control module at each router as it traverses the path; the essential point, from the perspective of keeping RSVP independent of the service model, is that RSVP itself does not make any modifications. Each switch maps this reservation request into the local service to be provided, and then makes the local admission control decision. If the reservation request is denied at a switch, the affected downstream receivers are sent an error message.[9] If the reservation request is successful at all switches along the path, no confirmation messages are sent.

In this one-pass approach, the reservation request cannot, in any reasonable manner, allow a receiver to specify an end-to-end delay or jitter bound. This is because the network cannot tell if the desired end-to-end service goals will be met by the resulting set of reservations. If the reservation request is not modified at each hop, then there is no way to ensure that the set of per-link services installed will meet any end-to-end service goals with nontrivial composition properties. If the reservation request is modified at each switch, each switch could install the minimal delay service and add their delay to a cumulative field in the reservation request so that the last switch along the path could check to see if the the requested delay bound was met. This, of course, is an extremely wasteful use of network resources and is not a viable approach.

Because the reservation request cannot reasonably specify the desired end-to-end service, in essence the reservation request can only meaningfully specify the per-link service desired. For instance, the request can specify that the receiver wants level 2 predictive service, or guaranteed service with rate r_g, at every hop (of course, the TSpec would also be specified). The receiver, however, never knows the end-to-end delays that result from the per-link service.

Thus, the one pass approach has the requester specify the

[9] Recall that RSVP reservation messages are receiver initiated. Therefore, error messages are sent downstream, to the receivers that sent the rejected reservation request.

per-link service to be reserved without knowing, either in advance or after-the-fact, the resulting end-to-end delays. This is inappropriate for either guaranteed or predictive service; if the receiver cannot ensure beforehand that an end-to-end delay bound will be satisfied, nor even find out what the resulting delay bound is after-the-fact, then there is no reason to offer delay bounds at all. Thus, the one-pass approach is only appropriate for the controlled delay service, which itself does not provide any quantitative characterization of its service.

The one-pass approach does have several attractive features. One important aspect is that it deals with multicast very well. By using the same distribution tree (traversed in the reverse direction) and merging reservation requests, the RSVP protocol manages to scale roughly as well as multicast routing does. In addition, the one-pass approach is idempotent; the reservation state present at the switch is the same as would be installed if the reservation request were resent. This enables RSVP to use a soft-state approach based on periodically resending the reservation requests to refresh the router state, which provides a simple and robust recovery mechanism. Lastly, the one-pass approach allows RSVP to be independent of the service model, in that the reservation requests are opaque blocks to the reservation protocol. It also allows the admission control module to be independent of RSVP, in that the system calls to admission control are merely service requests and they could be issued by any number of different protocols.

3.1.2 Two-Pass Mechanisms

Two-pass mechanisms, such as employed in the original Tenet architecture [1], have very different properties. We will first sketch this general approach as it applies to the unicast case. The receiver[10] sends a reservation message towards the source containing the desired end-to-end delay bound. Each switch makes a reservation with a tight delay bound on this first pass, and the cumulative delays along the path are recorded in the message. If, upon arriving at the source host, the cumulative delay in the reservation message is more than the desired end-to-end delay, then the source host issues a *reject* message and the initial reservations are released. If the cumulative delay in the reservation message is less than than the desired end-to-end delay, then the source host issues a *relax* message which carries information about how much excess delay there is (e.g., the extra delay per hop, or the total extra delay). As this return message passes through the switches, they *relax* their original reservations. Increasing the per-hop delays as much as possible while still meeting the end-to-end requirements enables the network to provide the requested service efficiently.

The two-pass approach enables receivers to meaningfully specify an end-to-end delay bound in their reservation requests. Thus, it can adequately support guaranteed and predictive service, unlike the simple one-pass approach. It can also trivially support controlled delay service, by not having any end-to-end delay target and having the first pass set up a given service level in each router.

However, the two-pass approach also has several drawbacks. Whether or not there is actual charging for network usage, users of the network should be aware of the "cost" of their usage in terms of resources consumed and opportunities denied other flows. When receivers specify a given end-to-end delay target, they do not know (either beforehand or after-the-fact) what per-link services must be installed. Receivers, therefore, do not know the "cost/performance" tradeoff along their path, and must issue their service requests in ignorance. If we believe that the delay targets are somewhat flexible, then we should allow receivers to make tradeoffs between performance and "cost".

Another problem with this two-pass approach is that it is not idempotent. The reserved state present at the router is not the same as would be installed if the reservation message were resent; the whole relaxation process would have to be restarted. This lack of idempotency makes it harder to adopt the soft-state approach. One could probably use a *triggered* relaxation process that was restarted only when a node detected that it was receiving a new reservation message (as opposed to a resent version of an old one), but it is hard to show that such an approach has no failure mode; the beauty of RSVP's soft-state approach is its simplicity.

Lastly, in such two-pass approaches, typically either the reservation protocol knows about the service model and performs the delay budget manipulations, or the admission control module knows that the reservation protocol is a two-pass protocol and can distinguish between the two passes. We think it better to have a cleaner division of labor.

A variation of the two-pass approach, which does not have some of the problems mentioned above is what we call the *passive* two-pass approach. The reservation requests specify the per-link services to be installed, and the initial pass is just like the original one-pass approach. The return pass no longer performs any relaxation. Instead, it is used merely to return to the receiver information about resulting end-to-end service. In the passive two-pass approach, the application specifies the per-link services and the network then informs the application of the resulting end-to-end service. This approach maintains the idempotent properties, and the clean separation between service model and reservation protocol, of the one-pass approach, while offering superior functionality. However, in contrast to the original two-pass approach, an application needing to achieve a specific delay target would have to iteratively search for the appropriate per-link request.

The above discussion only considered the unicast case, where the two-pass mechanism, both passive and otherwise, is fairly simple. While, the passive two-pass mechanism can be generalized easily to the multicast case, relaxation in the multicast case is somewhat harder. In particular, it seems impossible to achieve full relaxation when there are shared channels. While partial relaxation is still possible, using the two-pass-with-relaxation approach with shared channels leads to wasteful over-reservations. Because it is somewhat far afield from our current focus, we do not expand on this phenomena here except to note that for architectures which allow shared channels, it appears that only the passive two-pass approach is viable. In this vein, it is interesting to note that the revised Tenet approach (Suite 2) does not implement relaxation on the second pass.[11]

[10] For this discussion, it does not particularly matter whether the two-pass approach is sender-initiated or receiver-initiated in the unicast case; we describe the receiver-initiated version here.

[11] In this design a route server, which has knowledge of the path characteristics, determines the appropriate per hop services to

3.2 OPWA

We have discussed the features of the one-pass and two-pass approaches. We now propose a third alternative, which is a hybrid of the two previous approaches. It maintains the one-pass reservation messages in RSVP, but augments them with messages providing receivers with information about available services. We call it *one-pass with advertising* (OPWA).

3.2.1 Basic Mechanism

In the context of RSVP, this approach requires a new control message, which we call an ADV message. ADV messages are sent periodically from each source to all receivers.[12] Reservation requests are processed in a one-pass manner and specify the desired per-link service, just as they are in the current RSVP. The ADV messages are used to advertise (beforehand) the end-to-end service that would result from any given per-link service request. When going downstream, an ADV message carries with it the end-to-end delay bounds (both jitter and latency) that would result from reserving in various categories (see below for details). The receivers, knowing this information, can then request the level of service that best meets their needs. This is similar to the one-pass approach in that the receiver specifies the per-link service to be reserved rather than the end-to-end service. However, it is like the two-pass case in that the receiver knows what the resulting service will be *before* requesting service. Most importantly, this approach allows a receiver to evaluate the end-to-end delays resulting from various per-link service requests and then choose their desired level of service. This ability to scan the menu of available end-to-end services will allow the receiver to do a rough cost/performance tradeoff, which is not possible in the one-pass or two-pass approaches.

Advertisements are intended to provide receivers with information about the services offered along a path from the source. They do not indicate whether a particular service request would be admitted or rejected by admission control. Since the information provided in advertisements is fairly static, ADV messages can be refreshed at the same rate as other RSVP control messages.

3.2.2 Advertisements

The ADV messages contain fields relating to the end-to-end services available along that path. RSVP merely carries the ADV messages along but does not understand their content. These fields are updated by the admission control module at each hop. Each field is associated with a composition rule, which specifies how each router composes their local quantity into the quantity in the ADV message. To illustrate how this works, consider the delay bound for predictive service. There is a field in the ADV message for the end-to-end delay bound of each level of predictive service, call it D_i. When the ADV message arrives at a router α, the router inserts into this field the value $D_i + d_i^\alpha$, where d_i^α is the delay bound of predictive service level i at that link.

In general, each service-specific field is associated with local information used to update the field and a composition rule for how this information is combined with the information already present in the field; we will denote these composition rules in brackets. There are some generic advertised quantities contained in all ADV messages that are not associated with any specific service. For instance:

- Propagation delay [additive]
- Hop count [add 1]
- Bandwidth of links [minimum]

In addition, there are fields specific to the various services. For guaranteed service, the most relevant advertisements are:

- Error term A [additive]
- Error term B [additive]

These two fields, combined with the propagation delays above, enable a receiver to compute the end-to-end delays and jitter for their flow; this computation also involves the reserved rate r_g and the token bucket parameters (r, b), which the receiver already knows. (See Section 2 for definitions of A and B and for details of this computation).

For predictive service, the delay bounds are the obvious quantity to advertise. However, some applications would probably like to know the current delays as well. These current delays would most likely be some measure of the maximal delay recently seen, rather than a strict average of the delays. Thus, for each level i of predictive service the following quantities are advertised:

- Delay bound D_i [additive]
- Current delay C_i^p [additive]

For each level of controlled delay service, the only relevant advertisement is:

- Current delay C_i^{cd} [additive]

These quantities enable applications to determine what end-to-end service will result from a given service request. In the appendix we discuss various categories of applications and their likely use of this information. To make the presentation more concrete, we have described these advertisements in the context of a specific service model. The general approach applies to any service whose end-to-end characteristics can be computed on a sequential per-hop basis where the cumulative quantity in the ADV message is composed sequentially with the local per-hop quantity. We are not aware of any proposed service which does not fall into this category. Moreover, if the composition rule is associative and commutative, then the order in which the composition occurs is irrelevant.[13]

install.[17]

[12] RSVP currently has a message called a PATH message, which also travels from each source to all receivers. This message was originally intended mainly to install the appropriate routing state so that RSVP could run on top of almost any routing protocol, but may not be included in RSVP once routing protocols supply the appropriate functionality. A single message type can certainly embody the functionality of the current PATH message and the ADV message we propose here.

[13] This is important if other mechanisms for distributing the advertising contents are used, such as routing tables.

We are proposing advertising as a general paradigm. We described its use to advertise quantities relevant to the end-to-end quality of service. There are other measures of quality of service besides what we discussed here, such as measures of loss rates and convolutions of parametrized delay distributions. The advertising paradigm can be applied to other kinds of information as well. Routers can advertise their mean-time-between-failure (MTBF) and so applications can evaluate the likelihood of a failure along the path. In addition, advertising can be used to advertise the costs associated with link usage. This will be particularly useful in allowing applications to evaluate the cost/performance tradeoffs.

One can modify the semantics of reservation requests, while still retaining the OPWA model, to make better use of the advertising information. In the next two subsections we discuss two such modifications.

3.3 Achieving a Finer Delay Granularity

Because there are only three (or so) levels of predictive service, the delay targets at individual routers are likely to be widely separated. Thus, along a reasonably long path the differences in the end-to-end delays of the various predictive levels might be rather large (the difference along the path is the sum of the per-hop differences).[14] In this section we describe a way to enable users to request intermediate delays. Our point here is not to advocate this feature as much as illustrate how the information from ADV messages gives one the ability to implement new features by making the reservation request syntax more complex.

The reservation request for predictive service, in this case, will consist of a TSpec, a predictive service level, and a scalar E. E represents the extra delay that can be absorbed by the network along the path; i.e., if the end-to-end delay in the requested predictive service class is D, the application is indicating that it is willing to tolerate end-to-end delays up to $D + E$. When the request arrives at a router the reservation can be made for a lower class (i.e., higher delay) of predictive service than that requested if the extra delay is less than E; if the reservation is made for the lower class, the E field is decremented by the difference.[15] The greedy algorithm (where you use up as much of E as possible at each hop) leads to making low quality reservations near the receiver and keeping high quality reservations closer to the source. When sharing among different receivers, this is exactly the desired behavior of sharing the high quality "expensive" reservations.

3.4 Shared Channels

Consider a shared channel reservation for an audio teleconference. There is a single reserved "pipe" used by all sources, going to all receivers. It would be natural that each receiver would want a reservation where the end-to-end delay to each sender is roughly the same. This cannot be done with reservation requests that merely specify a single level of service for the reverse distribution tree.[16] Specifying a single level of service on all links would lead to unnecessarily stringent reservations on some links. Relaxing the service on those links will provide more efficient service while still meeting the application's requirements. To achieve this goal, the per-hop service installed on a link must depend on exactly which sources are upstream. We could define a reservation request of the form:

```
If source S1 is upstream, reserve at least R1
If source S2 is upstream, reserve at least R2
......
If source Si is upstream, reserve at least R3
......
```

where $R1$, $R2$, $R3$ are per-hop service requests with a natural ordering among them (e.g., they are all levels of predictive service). Thus, at every hop, the router can determine the necessary service because it knows, from multicast routing, the set of upstream sources. The receiver can compute the appropriate values for the Ri from the advertising information.

This clearly does not scale because of the need to list all the sources. The RESV messages sent over a link are as big as the number of sources upstream (as the RESV messages go upstream, entries that are no longer needed can be discarded). However, recent discussions in the RSVP design community suggest that RSVP itself, in order to support shared channels, might require RESV messages to include such a list of sources, at least when running on top of current multicast routing protocols [2].

3.5 Interactions with Type-of-Service Routing

There are several proposals for routing algorithms that can provide paths tailored for an application's service requirements; see [4, 9, 20] for a few representative references. These protocols offer an application a choice of path, and OPWA offers an application a choice of services along a path. How does OPWA interact with such routing algorithms?

These routing algorithms choose routes based on a *Quality-of-Route* (QoR) request. The basic assumption we make is that QoR is measured on a much coarser granularity than the differences in service provided by the different service classes along a given route. While such routing algorithms will likely supply routes that minimize latency, or of maximize bandwidth, we do not think Type-of-Service routing will ever support a request of the form: "give me a route with 100 millisecond delay in level 2 predictive". Controlling the delay at this fine level of granularity is the job of packet scheduling, and should be done through the reservation requests.

To be more specific, when joining a multicast group a receiver might include, in its join message, an indication of its desired QoR, such as delay sensitivity and likely bandwidth requirements (order of magnitude). Routing finds a route, and then ADV messages start flowing down the path. The receiver can then evaluate the offered services and decide if they are sufficient. If not, the receiver can ask routing for

[14] Note that guaranteed service does not have this problem since the receiver can continuously tune the resulting delays by varying r_g.

[15] The mapping of request to service at the router must be independent of the other reservations present; otherwise one ends up with pathological cases where when another flow leaves your reservation request (which had been granted) is suddenly denied.

[16] This is a reverse tree because the tree consists of the paths from all sources down to this single receiver.

an alternate route, and that alternate route request could specify a different QoR.

Thus, Type-of-Service routing and OPWA are complementary because they operate at rather different granularities of service. Of course, there are many other interesting interactions between routing and reservation protocols; see [10] for a more extensive discussion.

3.6 Discussion

The *philosophy* of OPWA is that receivers are given enough information so that they know the end-to-end service that would result from any per-link service request. This allows applications to make tradeoffs between "cost" and performance. Moreover, because the reservation establishment process itself remains one-pass, the various mechanisms are quite simple and robust.

This philosophy can be implemented in many ways. The OPWA *mechanism* we presented here has two basic aspects. The first, and more fundamental, aspect of this mechanism is that routers make available through some general interface information about the services they provide. We refer to this as *advertising*. The second aspect of the mechanism is that ADV messages collect these advertisements along the path and deliver them to receivers. There are other ways to accomplish this same goal; for instance, static advertised parameters could be made available through routing.[17] In particular, some architectural proposals for the Internet, such as Nimrod [4], provide link-state maps to each end host. Certainly in such an architecture, the ADV messages we propose here would be largely superfluous (they would still be needed to carry any advertisements that varied on time scales shorter than the frequency of routing updates). However, routing does not provide such information now, nor do we expect it to do so before the real-time extensions to the service model are adopted. Thus, we must propose a solution that does not rely on routing to provide these quantities.[18]

OPWA was presented here in the context of a specific service model. OPWA is in no way tied to the details of this service model, and OPWA should not be confused with the particular service model we used as an example. If, for instance, the delay bounded services used other composition rules to calculate their end-to-end delay, OPWA could be used with them as well (as long as the end-to-end delay can be computed by a sequential computation along the path). OPWA is not needed if all services have trivial composition rules, and so the only necessary property of the service model, as far as our discussion here is concerned, is that there be some services with nontrivial composition rules, and that these composition rules allow one to compute the end-to-end service from a sequential pass along the path.

[17] The Tenet architecture, in Suite 2, has adopted a very similar approach in that its route servers have complete path information. However, it is the route servers which do the translation from desired end-to-end service to required per-link service, whereas in OPWA the receiver does the translation.

[18] It is debatable whether routing should ever provide these quantities, but we do not pursue that here. It suffices to observe that routing will not provide those quantities in the near future.

4 Heterogeneity

If one were designing and deploying a new network from scratch, and did not expect its service model to evolve, then it might be reasonable to assume that all routers and subnets supported every service in the service model. When designing for the existing Internet, which is large and administratively decentralized, and where one expects future extensions to the service model, such homogeneity is impossible to achieve. Whenever new services are added to the service model, deployment of these new services in network routers will take time, and during the transitional period some routers will support the new services while others will not. Upgrading or replacing existing infrastructure to support new services will be a gradual process. Moreover, implementing the algorithms required to offer real-time service will be difficult or impossible on certain network technologies, such as ethernet, so real-time services may never be deployed on such subnets. Therefore, heterogeneity is inevitable in the Internet, and we must incorporate its presence in our design.[19] New services can be standardized for the Internet, but their deployment can not be mandated. Whether or not a service is deployed remains under the control of individual network service providers and router vendors. Different service providers may choose to deploy a different subset of newly defined services. Thus, different real-time services will be available at different routers in the Internet, and some routers may not support any real-time service at all.

The resulting heterogeneity implies that the set of services available end-to-end will depend on the particular path, and routers along a given path may not all support the same services. Without mechanisms to address this heterogeneity, routers not supporting a service requested by an application must deny the service request. This enables applications to use only those services offered by all routers along a path. This restriction to the "best common service" significantly limits the services available to applications. For example, applications running on any host attached to an ethernet would be unable to secure anything other than best-effort service at all routers along a path. Moreover, such a design would severely inhibit the incremental deployment of new services and the use of partially deployed ones. This default "best common service" behavior would therefore greatly slow, and perhaps even prevent, the evolution of the Internet towards an integrated services architecture. Thus, mechanisms that address the problem of heterogeneity are a necessary component of an integrated services architecture.

4.1 Replacement Services

The approach we take to heterogeneity is based on the principle of *replacement* services. When a particular service is not offered end-to-end, it can be replaced at one or more routers by an alternate service. The fundamental principles underlying this approach are that (1) the service expected

[19] Indeed, the Integrated Services Working Group of the IETF is currently operating under the assumption that while the IETF will define and standardize a service model like the one described in Section 2, it cannot mandate the universal deployment of the constituent services. See the archives of the Integrated Services Working Group, available for anonymous ftp on ftp.isi.edu, for a further discussion of this subject. In particular, see [27].

by an application should be modeled on the ideal of end-to-end connectivity of the requested service (i.e., the service that would be provided if all routers along the path offered the requested service), and (2) the network is responsible for informing the application the extent to which this is a realistic approximation. This insulates applications from having to understand, or even know about, the detailed nature of heterogeneity or how the different services along the path compose. This insulation is critical, we feel, for enabling applications to function relatively undisturbed while the network infrastructure evolves.

The use of replacement services can be viable for two reasons.[20] First, local conditions may enable a router to substitute one service for another without a perceptible difference in the quality of service delivered to the application. For example, a lightly loaded ethernet that offers only best-effort service may have low enough delays that it provides service as good as is required by predictive service. Second, applications vary in the degree of assurance they require about the service they receive. An adaptive application that is tolerant of packet loss, may be satisfied with best-effort service at those routers where real-time service is not available.

We characterize replacement services as *reliable* or *unreliable*. A reliable replacement is one that meets the service specifications of the original service a large majority of the time. We do not express the degree of compliance precisely, but for purposes of the present discussion we assume reliable replacements achieve this conformance well over 95% of the time.[21] While this lack of specification may seem vague, we assume that over time operational guidelines and informal standards of acceptable practice will emerge. The key point behind the notion of reliable replacements is an implicit statement that use of the replacement will result in service that is usually not perceptibly different from the requested service. Examples of reliable replacements could be the use of predictive service for guaranteed service, or the use of best effort service on an underutilized ethernet as a replacement for controlled delay service.[22]

Unreliable replacements are a weaker form of replacement services, carrying no assurance about the quality of the resulting service. That is, service offered as an unreliable replacement can be arbitrarily bad. For example, controlled delay service or best-effort service on an often congested network could be considered unreliable replacements for guaranteed service. Some applications may be willing to use unreliable replacements when no alternative exists, since poor service will in some cases be better than no service at all.

The use of replacements (reliable and unreliable) leaves it to the router to decide which service best fits the original request, and whether or not this substitute constitutes a reliable replacement. This allows overprovisioning of less stringent services to serve as replacements for more demanding ones. A network built around overprovisioned best-effort service cannot claim to support guaranteed or predictive service, but it could claim to provide reliable replacements for them. In some cases users will care about the difference in perceived quality between guaranteed or predictive service and its reliable replacement and will not use reliable replacements, but for the majority of cases the difference in perceived quality will not matter.[23]

4.2 Advertising Service Availability

In our approach to dealing with heterogeneity of service offerings, routers make known the availability of services and their replacements via the ADV messages introduced in Section 3. While routers may decide whether or not to offer particular services, they must advertise the availability (or lack thereof) of every service. For each service, the router advertises one of the following four conditions:

1. Offered.
2. Reliable replacement available.
3. No reliable replacement available.
4. Not known.

The first condition indicates the router offers the service as defined in the service model. The second indicates the router does not offer the service, but instead offers a reliable replacement. The third indicates that the router offers neither the actual service nor a reliable replacement. Every router implicitly offers an unreliable replacement, as best-effort service (which all routers must offer) constitutes an unreliable replacement for all services. Finally, "not known" means that in addition to not offering the specified service, the router does not even know what the service is. For instance, a newly defined service will not be known to most deployed routers. In contrast, known but not offered services (the second and third advertisements) are those that the router knows about but chooses not to offer.

For each service, the advertisements keep a count of how many routers fall into each category (offered, reliable replacement available, etc.). Thus, a setup protocol like RSVP

[20] Routers are always free to substitute a "better" service, in the sense of the ordering used for merging (which we discussed in Section 2.3). For instance, a router can always use a higher lever of predictive service than the one requested, and in fact merging depends on such substitutions. Less trivially, if predictive is always considered better than controlled delay in the ordering relationship, then a router can always substitute predictive service for controlled delay service. We do not use the term replacements to refer to such "ordered substitutions".

[21] There is a subtle distinction between the notion of reliability in predictive service and the reliability of a "reliable replacement." For a router to offer predictive service, it must be able to meet the service requirements independent of where it is deployed and the ambient load. Adherence to the service requirements could be evaluated with conformance testing, in which a router is subjected to a wide variety of loads and its ability to deliver the advertised service is measured. In contrast, the reliability of a reliable replacement need only apply to the particular context where assumptions can be made about the nature of the load. Thus, an overprovisioned router with FIFO scheduling, and no admission control, does not support predictive service, but can offer a reliable replacement for predictive service.

[22] The specific examples given of replacement services, and their classification as reliable or unreliable, should not be taken as a definitive statement about the suitability of these services to act as replacements. This ultimately depends on the specific services defined and on decisions taken by routers, which can be impacted by local conditions.

[23] There is a vigorous debate in the Internet community about whether the overprovisioning of best-effort links is a reasonable real-time architecture. Our proposal would allow such overprovisioned best-effort networks to compete with networks supporting real-time services. The community can thus experiment with different ways of meeting application needs without changing the service model. We think this is crucial in allowing us to learn as we build the next generation Internet.

can use advertising to transmit to the application the degree of service availability end-to-end. Replacements can only be used if applications know about their use; that is, the only reasonable semantics for a reservation request, in the absence of other information (i.e., that providing by advertising), is that the application should assume they received the service they asked for unless they are sent a notification of failure. ADV messages enable us to provide notification of replacements to applications beforehand. An application that would not be satisfied with the resulting service has the option to not request the service. For example, an application that requires a fairly reliable assurance about the service it receives might not request service if one or more routers advertises no reliable replacement. Hence, advertising permits applications to avoid end-to-end service that does not meet their needs. An enhanced routing architecture might be employed to find alternate routes which do meet the end-to-end service requirements in such a circumstance.

At each hop, the router attempts to update the advertisement fields with meaningful values. When a replacement service is offered (reliable or otherwise), the router fills in the advertisement in the normal manner. For a reliable replacement, the advertised values should accurately reflect the service provided by the replacement service. A router can also attempt to fill in an advertised value for an unreliable replacement since it understands something about the service. For example, a router that uses an unreliable replacement and is consequently unable to guarantee packet delivery might still be able to advertise an estimated delay bound based on delays experienced at the router. However, advertisements for unreliable services will often be of little value. Finally, routers cannot advertise anything meaningful for services that are not known since they cannot understand the rules for updating the advertised values. For example, a router cannot know whether the advertised value for an unknown service reflects the delay experienced end-to-end or some other value. Applications receiving advertisements will know how much confidence to place in the advertised values based on the availability of the services end-to-end.

4.3 Router Substitutions

Requests for services not offered by a router can be divided into two categories: services known but not offered by the router and services that are neither offered nor known. In the former case the router substitutes its best replacement service for the original request. This means a reliable replacement is substituted if one is offered, and an unreliable replacement is substituted if no reliable replacement is offered. The service request is then processed as if it were a request for the replacement service. If the replacement service requires admission control, the router uses the admission control algorithm to determine the admissibility of the new request. The service request is rejected if the admission control test fails; substitutions of additional replacement services are not permitted because the resulting service would not conform to the previously advertised values.[24] If the service request is admitted, or if the replacement service does not require admission control, the original service request is forwarded to the next router along the path.

When a router receives a service request for an unknown service, it cannot even judge the suitability of services as replacements. The best it can do is to substitute the ubiquitously available best-effort service. Since the router cannot update the ADV fields for unknown services, any advertising information pertaining to this service will be of little use to the application.

We show in some simple examples how advertising and service replacements can be used to deal with heterogeneity. We expect the two scenarios we describe here to be representative of situations likely to be encountered in the future Internet. Consider first a situation in which a single real-time service (e.g. guaranteed) is offered at some routers along a path, and other routers offer only best-effort service. These routers will substitute best-effort service for the request for guaranteed service. Thus, the use of service replacements allows an application to obtain the real-time service where it is offered. Applications are not confined to using a common service offered end-to-end. Instead, the end-to-end service is built out of heterogeneous services offered at the routers.

Next, we consider the case in which all routers offer real-time service, but they do not all offer the *same* real-time service. For example, some routers may offer guaranteed service, while others offer predictive (but neither service is offered at all routers). Without a mechanism to allow for the composition of end-to-end service out of different services, applications would only be able to use best-effort service (which is offered everywhere). In this example, if routers use the real-time service they offer as a replacement for the service they do not offer, applications can acquire a real-time service at each router. For example, an application may request guaranteed service, and routers that offer only predictive can substitute predictive. Using these replacements, applications receive better service than if they were confined to using a single service.

The previous example depends on routers using the real-time services they offer as replacements for ones they do not offer. While the intelligent use of replacements at routers cannot be mandated, we believe that service providers will have strong incentives to use the best replacement strategies they can. Offering replacement services will make their services more usable and will therefore increase the utilization of their networks. Note that in the above examples, advertising provides applications with knowledge about end-to-end service offerings. Therefore, an application that requires a particular service at every router has the option not to issue a service request when such a request would result in the substitution of replacements for the requested service.

4.4 Other Approaches

To put our solution in context, it is worth mentioning alternative approaches to solving this problem which we considered. Another way that heterogeneity could be addressed is to allow applications to control router actions when the desired service is not offered. Abstractly, a reservation request could contain a *reservation script* that specifies cer-

[24] We do not address the issue of allowing substitutions for admission control failures here, as it is not specifically a function of heterogeneity in service availability. Admission control failures will occur even when all routers offer the same services. Hence, the question must be considered in the context of a wider discussion of the integrated services architecture. However, we do think this is an important issue, especially when admission control failures are a result of merging problems (i.e., a predictive request and a guaranteed request not being able to merge).

tain router actions if the requested service is not available. For example, an application could express a preference for the router not to use any replacements, to use only reliable replacements, or to use any replacement service. This approach obviates the need for the availability and reliability of replacement services to be included in advertising messages. However, we see little benefit to this tradeoff, and the use of reservation scripts introduces a very serious merging problem. Specifically, if applications can express their desires through individualized reservation scripts, the presence of multicast flows implies that two receivers of the same flow may specify conflicting scripts. One receiver may want any available replacement to be used, while another may not want service consisting of replacement services. What action should a router that must merge these conflicting scripts take? The correct behavior is to give applications only those replacement services they are willing to accept, while not denying them access to acceptable replacements. Achieving this would significantly increase the complexity of a reservation protocol.

One can extend the idea of reservation scripts to have individual applications indicate specifically which services should be used as replacements. This idea continues to have the same merging problem described above. In addition, and perhaps more importantly, we believe it places the responsibility for replacements in the wrong hands. Since the quality of service provided by best-effort and controlled delay services will likely depend a great deal on the ambient load conditions, the local router is much better equipped to make the decision about whether or not a given service can function as a reasonable replacement (if the router knows about the service). There may be a few exceptional cases where a particular application requirement would suggest a different substitution, but we are not convinced that such occurrences outweigh the considerable additional complexity such specific reservation scripts would introduce.

A more limited but similar alternative we considered is to include an explicit list of services in the reservation request. Upon receiving a reservation request, a router would scan the list of requested services until finding the first one that it knows about. It would then process the request as a request for this service. If this service is known, but not offered, the router would substitute a replacement; it does not continue to scan the list of services for one it offers. Thus, receiver specified replacement services are used only when the router does not know about the receiver's first choice service; in all other cases, the decision about replacement services is left to the router. Since it is specifically in situations where services are not known that routers lack sufficient information to make intelligent decisions about replacement services, this limited control exerted by the receiver might be useful.

We conclude this section with a statement about the generality of our proposed approach. We believe that heterogeneity will be an issue, regardless of what service model is defined for real-time services. Even if only a single type of real-time service were to be specified, this service will not be universally deployed. Some routers will offer it and others will not. However, we expect that the requirements of real-time applications differ enough to warrant multiple types of real-time service, each geared towards supporting different classes of applications. Multiple classes of service, none of which will be offered everywhere, only serves to compound the problem of heterogeneity.

5 Discussion

In this paper, we considered two basic issues in reservation establishment: allowing applications to control the end-to-end service their flows receive, and coping constructively with heterogeneity in service offerings. Any architecture that does not address these two issues is likely to fail. The one-pass-with-advertising approach advocated here enables applications to peruse the menu of possible end-to-end services before submitting their service requests. In addition to giving applications the information they need about their service requests, one-pass-with-advertising does not violate the principle of idempotency needed to implement the soft-state approach. The use of replacement services, in combination with the advertisement of service availability, provides the semblance of homogeneity in a heterogeneous environment in that applications do not have to deal directly and explicitly with the variety of router capabilities along the path.

We have provided a basic architectural description of these proposals. While we have not included the details required for an actual protocol specification, the basic mechanisms are straightforward. Adding advertising messages to a reservation protocol such as RSVP should not be difficult, and the mechanisms for heterogeneity should require fairly simple changes to the admission control modules of a router offering real-time services. Beyond that, the crucial questions about this design revolve around the nature of future applications and the degree of heterogeneity in future networks. For example, do applications really need to know about the resulting service beforehand? Are applications willing to tolerate an end-to-end path consisting of heterogeneous per-link services? Is the ability for an application to choose from an advertised set of services and corresponding performance values useful? We believe the answer to these questions is yes, but only through implementation of our proposed enhancements and experimentation with actual applications that utilize these added features will these questions be definitively resolved.

In this paper, we addressed what we considered are the two most important open issues in the design of a reservation protocol. However, other interesting questions remain. In Section 4, we alluded to the possibility of service substitution on admission control failures. This is one issue that arises in the context of a larger question. Namely, can an application that is willing to accept any of several services offered by the network make a service request reflecting this flexibility? From the application viewpoint, a reservation protocol and service model that can support this functionality is desirable. Whether or not this functionality is possible without adding an unacceptable degree of complexity remains to be seen.

Acknowledgements

This paper reflects insights gleaned from ongoing discussions with our many other colleagues in the Integrated Service Internet Project – which is an informal collaboration involving researchers from Xerox PARC, MIT, USC and ISI – and the Integrated Services and RSVP Working Groups of the IETF. We would specifically like to thank Steve Berson, Bob Braden, David Clark, Steve Deering, Deborah Estrin, Sally

Floyd, Shai Herzog, Sugih Jamin, Danny Mitzel, Craig Partridge, John Wroclawski, Daniel Zappala, and Lixia Zhang. We would also like to thank Amit Gupta and Srinivasan Keshav for several helpful conversations.

References

[1] A. Banerjea, D. Ferrari, B. Mah, M. Moran, D. Verma, and H. Zhang. *The Tenet realtime protocol suite: design, implementation, and experiences*, Technical Report TR-94-059, International Computer Science Institute, November 1994.

[2] S. Berson and D. Zappala. *Looping and Wildcard Filters*, memo 1/26/95.

[3] R. Braden, D. Clark, and S. Shenker. *Integrated Services in the Internet Architecture: an Overview*, **Internet RFC 1633**, June, 1994.

[4] I. Castineyra, J.N. Chiappa and M. Steenstrup. *The Nimrod Routing Architecture*, Internet Draft, 1994.

[5] D. Clark. *The Design Philosophy of the DARPA Internet Protocols*, In **Proceedings of ACM SIGCOMM '88**, August 1988.

[6] D. Clark, S. Shenker, and L. Zhang. *Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism*, In **Proceedings of ACM SIGCOMM '92**, pp 14-26, 1992.

[7] S. Deering. *Multicast Routing in a Datagram Internetwork*, Technical Report STAN-CS-92-1415, Stanford University, December 1991.

[8] A. Demers, S. Keshav, and S. Shenker. *Analysis and Simulation of a Fair Queueing Algorithm*, In **Journal of Internetworking: Research and Experience**, 1, pp. 3-26, 1990. Also in Proc. ACM SIGCOMM '89, pp 3-12.

[9] D. Estrin, Y. Rekhter and S. Hotz. *Scalable Inter-Domain Routing Architecture*, In **Proceedings of ACM SIGCOMM '92**, August 1992.

[10] D. Estrin, S. Shenker, and D. Zappala. *Routing Support for Resource Reservations*, preprint, 1994.

[11] D. Ferrari. *Distributed Delay Jitter Control in Packet-Switching Internetworks*, In **Journal of Internetworking: Research and Experience**, 4, pp. 1-20, 1993.

[12] D. Ferrari, A. Banerjea, and H. Zhang. *Network Support for Multimedia: a discussion of the Tenet approach*, In **Computer Networks and ISDN Systems**, 20, pp. 1267-1280, 1994.

[13] J. Forgie. *ST – A Proposed Internet Stream Protocol*, Internet Experimental Notes IEN-119, September 1979.

[14] I. Gopal and R. Guérin. *Network Transparency: The plaNET Approach*, In **IEEE/ACM Transactions on Networking**, 2(3), pp. 226-239, 1994.

[15] A. Gupta and M. Moran. *Channel Groups: A Unifying abstraction for specifying inter-stream relationships*, Technical Report TR-93-015, International Computer Science Institute, March 1993.

[16] A. Gupta, W. Howe, M. Moran, and Q. Nguyen. *Resource sharing in multi-party realtime communication*, In **Proceedings of IEEE Infocom '95**.

[17] A. Gupta, private communication.

[18] J. Hyman, A. Lazar, and G. Pacifici. *A Separation Principle Between Scheduling and Admission Control for Broadband Switching*, In **IEEE JSAC**, Vol. 11, No. 4, pp 605-616, May 1993.

[19] C. Kalmanek, H. Kanakia, and S. Keshav. *Rate Controlled Servers for Very High-Speed Networks*, In **Proceedings of GlobeCom '90**, pp 300.3.1-300.3.9, 1990.

[20] J. Moy. *OSPF Version 2*, **Internet RFC 1583**, March 1994.

[21] A. Parekh and R. Gallager. *A Generalized Processor Sharing Approach to Flow Control- The Single Node Case*, In **IEEE/ACM Transactions on Networking**, 1(3), pp. 344-357, 1993.

[22] A. Parekh and R. Gallager. *A Generalized Processor Sharing Approach to Flow Control- The Multiple Node Case*, In **IEEE/ACM Transactions on Networking**, 2(2), pp. 137-150, 1994.

[23] A. Parekh. *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks*, In **Technical Report LIDS-TR-2089**, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1992.

[24] H. Saran, S. Keshav, and C. Kalmanek. *A Scheduling Discipline and Admission Control Policy for Xunet 2*, **NOSSDAV '93**.

[25] S. Shenker, D. Clark, and L. Zhang. *A Scheduling Service Model and a Scheduling Architecture for an Integrated Services Packet Network*, preprint.

[26] S. Shenker. *Fundamental Design Issues for the Future Internet*, In **IEEE JSAC**, to appear.

[27] S. Shenker. mail to int-serv@isi.edu mailing list, 11/4/1994.

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

[29] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. *RSVP: A New Resource ReSerVation Protocol*, **IEEE Network Magazine**, September 1993.

[30] L. Zhang, R. Braden, D. Estrin, S. Herzog, and S. Jamin. *Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification*, RFC in preparation, 1995.

A Application Requirements

In this appendix we present a slightly more general formulation of the issue addressed in Section 3. That is, rather than asking merely how to achieve end-to-end delay bounds, we ask: what information do applications need both before and after making reservations? To address this question, we use the following taxonomy of applications. We do not claim that this taxonomy is all-encompassing, nor that its distinctions are sharp, nor that all categories we list are nonempty; we merely find the taxonomy useful in guiding the following discussion.

We will assume that the bulk of real-time applications are *playback* applications.[25] It is useful to distinguish between four broad categories of playback applications. This taxonomy is a generalization of that presented in [6, 25]. *Intolerant* playback applications cannot accept any violations of the delay bound; these applications must request guaranteed service and set their playback point to the maximal delay specified by the delay bound. *Tolerant* applications, on the other hand, can tolerate occasional violations of delay bounds. Some tolerant applications do not adapt to the actual delays of their packet and, like intolerant applications, set their playback point to the delay bound; this lack of adaptation need not be because of the inability of the application to adapt, but rather because the application cares more about fidelity than about reduced delay. For example, when listening to a speech, or a concert, the delays are typically less important than fidelity. These *tolerant but not-adapting* applications would presumably request predictive service. However, many tolerant applications will adapt to the actual received delays, and these applications need not set their playback point to the given delay bound. We call these *tolerant delay-adaptive* applications, and they would presumably request either predictive or controlled delay service. For the previous three application classes we have implicitly assumed that they request a given fixed level of service, and continue to use that level of service for the duration of the flow. *Tolerant service-adaptive* applications not only adapt to current delays by changing their playback point, but they also change their service request to the appropriate level depending on the received delays. This implies a greater degree of tolerance than is needed for delay-adaptation, and the appropriate service class is either predictive or, more likely, controlled delay service.

Of course, another distinction between applications, quite orthogonal to the ones above, is between interactive and noninteractive applications. Interactive applications such as a teleconference require low end-to-end delays, whereas noninteractive applications such as a broadcast movie typically do not. Adapting to the delivered delays, in order to take advantage of the fact that the delivered delays are typically much smaller than the delay bounds, is really only an advantage for interactive applications. Noninteractive applications would only be adaptive out of necessity; that is, if they knew the delay bound of their incoming packets, they would be just as happy to set the playback point at this bound.[26]

The classes of applications described above differ not only in the services they might request, but in the information they need during the reservation process. We now briefly address two questions: (1) what information does an application need before making a reservation and (2) what information does an application need after making a reservation?

[25] Certainly there are some real-time applications which are not playback applications, but the discussion that follows applies to them as well.

[26] We are assuming, for convenience, that the buffer requirements are not much of an issue. If they are, then noninteractive applications might become adaptive in order to decrease their buffer consumption.

What information does an application need before making a reservation? Interactive applications typically have some delay requirements that can be expressed by a delay target. For the various application classes, this target delay may need to be a guaranteed delay bound (intolerant nonadaptive), a predictive delay bound (tolerant but-not-adapting), or an estimate of the likely delays (tolerant adaptive). Only service-adaptive applications need not know the target delays beforehand, because they are the only ones prepared to switch between service levels. Note, however, that not knowing anything about the likely delays beforehand means that service-adaptive applications are risking unknown and perhaps substantially worse delays whenever they lower their requested service level. It would be helpful, though perhaps not necessary, for even these applications to know something about the delivered delays in advance. Thus, before requesting service, most interactive real-time applications would like to know, at least approximately, the likely delays and/or jitters (either experienced or bounds) that would result from a given reservation request.

What information does an application need after making a reservation? Applications which are not adapting to the delivered delays (either out of design or out of choice) need to know the resulting end-to-end delay bounds in order to safely set the offset delay. Adaptive applications do not need to know the resulting end-to-end delay bounds, although knowledge of a delay bound might help in the adaptation process, and in setting aside buffers.

Thus, the information provided by the traditional one-pass approach is inadequate unless all applications are tolerant and service-adaptive. Even then, one-pass is sufficient only if these applications are willing to sample services without any pre-knowledge of the likely resulting service.

We make this point because it is often claimed that adaptive applications make it unnecessary to provide information about the service being provided. It is true that adaptive applications do not need know the delay bounds to *use* the service; they can set their playback point themselves. But unless applications are service-adaptive as well, and can tolerate periods of substantially longer delays, then interactive applications need this information to *choose* the service level. The information relevant to this choice is probably not the delay bound, or at least not only the delay bound, but there is a need for some knowledge beforehand. This is why we mentioned advertising currently-experienced-delay, and doubtless other such quantities will be proposed.

The widespread use of *nv*, *vat*, and other similar tools on the Internet has given us some experience with delay-adaptive applications. However, we have little experience with service-adaptive applications. We think it somewhat premature to base an architecture on the hope that service-adaptive applications will become dominant, and thus we think OPWA, or some equivalent approach, is a necessary component to the future Internet.