Network Working Group
Request for Comments: 1633
Category: Informational

R. Braden
ISI
D. Clark
MIT
S. Shenker
Xerox PARC

July 1994

Integrated Services in the Internet Architecture: an Overview

Status of this Memo

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Abstract

This memo discusses a proposed extension to the Internet architecture and protocols to provide *integrated services*, i.e., to support real-time as well as the current non-real-time service of IP. This extension is necessary to meet the growing need for real-time service for a variety of new applications, including teleconferencing, remote seminars, telescience, and distributed simulation.

This memo represents the direct product of recent work by Dave Clark, Scott Shenker, Lixia Zhang, Deborah Estrin, Sugih Jamin, John Wroclawski, Shai Herzog, and Bob Braden, and indirectly draws upon the work of many others.

Contents

1 Introduction

The multicasts of IETF meetings across the Internet have formed a large-scale experiment in sending digitized voice and video through a packet-switched infrastructure. These highly-visible experiments have depended upon three enabling technologies. (1) Many modern workstations now come equipped with built-in multimedia hardware, including audio codecs and video frame-grabbers, and the necessary video gear is now inexpensive. (2) IP multicasting, which is not yet generally available in commercial routers, is being provided by the MBONE, a temporary "multicast backbone". (3) Highly-sophisticated digital audio and video applications have been developed.

These experiments also showed that an important technical element is still missing: real-time applications often do not work well across the Internet because of variable queueing delays and congestion losses. The Internet, as originally conceived, offers only a very simple quality of service (QoS), point-to-point *best-effort* data delivery. Before real-time applications such as remote video, multimedia conferencing, visualization, and virtual reality can be broadly used, the Internet infrastructure must be modified to support *real-time* QoS, which provides some control over end-to-end packet delays. This extension must be designed from the beginning for multicasting; simply generalizing from the unicast (point-to-point) case does not work.

Real-time QoS is not the only issue for a next generation of traffic management in the Internet. Network operators are requesting the ability to control the sharing of bandwidth on a particular link among different traffic classes. They want to be able to divide traffic into a few administrative classes and assign to each a minimum percentage of the link bandwidth under conditions of overload, while allowing "unused" bandwidth to be available at other times. These classes may represent different user groups or different protocol families, for example. Such a management facility is commonly called *controlled link-sharing*. We use the term *integrated services* (*IS*) for an Internet service model that includes best-effort service, real-time service, and controlled link sharing.

The requirements and mechanisms for integrated services have been the subjects of much discussion and research over the past several years (the literature is much too large to list even a representative sample here; see the references in [CSZ92, Floyd92, Jacobson91, JSCZ93, Partridge92, SCZ93, RSVP93a] for a partial list). This work has led to the unified approach to integrated services support that is described in this memo. We believe that it is now time to begin the engineering that must precede deployment of integrated services in the Internet.

Section 2 of this memo introduces the elements of an *IS* extension of the Internet. Section 3 discusses real-time service models [SCZ93a, SCZ93b]. Section 4 discusses traffic control, the forwarding algorithms to be used in routers [CSZ92]. Section 5 discusses the design of RSVP, a resource setup protocol compatible with the assumptions of our *IS* model [RSVP93a, RSVP93b].

2 Elements of the Architecture

The fundamental service model of the Internet, as embodied in the *best-effort* delivery service of IP, has been unchanged since the beginning of the Internet research project 20 years ago [CerfKahn74]. We are now proposing to alter that model to encompass integrated service. From an academic viewpoint, changing the service model of the Internet is a major undertaking; however, its impact is mitigated by the fact that we wish only to *extend* the original architecture. The new components and mechanisms to be added will supplement but not replace the basic IP service.

Abstractly, the proposed architectural extension is comprised of two elements: (1) an extended service model, which we call the *IS* model, and (2) a reference implementation framework, which gives us a set of vocabulary and a generic program organization to realize the *IS* model. It is important to separate the service model, which defines the externally visible behavior, from the discussion of the implementation, which may (and should) change during the life of the service model. However, the two are related; to make the service model credible, it is useful to provide an example of how it might be realized.

2.1 Integrated Services Model

The *IS* model we are proposing includes two sorts of service targeted towards real-time traffic: guaranteed and predictive service. It integrates these services with controlled link-sharing, and it is designed to work well with multicast as well as unicast. Deferring a summary of the *IS* model to Section 3, we first discuss some key assumptions behind the model.

The first assumption is that resources (e.g., bandwidth) must be explicitly managed in order to meet application requirements. This implies that *resource reservation* and *admission control* are key building blocks of the service. An alternative approach, which we reject, is to attempt to support real-time traffic without any explicit changes to the Internet service model.

The essence of real-time service is the requirement for some service guarantees, and we argue that guarantees cannot be achieved without reservations. The term "guarantee" here is to be broadly interpreted; they may be absolute or statistical, strict or approximate. However, the user must be able to get a service whose quality is sufficiently predictable that the application can operate in an acceptable way over a duration of time determined by the user. Again, "sufficiently" and "acceptable" are vague terms. In general, stricter guarantees have a higher cost in resources that are made unavailable for sharing with others.

The following arguments have been raised against resource guarantees in the Internet.

- "Bandwidth will be infinite."

  The incredibly large carrying capacity of an optical fiber leads some to conclude that in the future bandwidth will be so abundant, ubiquitous, and cheap that there will be no communication delays other than the speed of light, and therefore there will be no need to reserve resources. However, we believe that this will be impossible in the short term and unlikely in the medium term. While raw bandwidth may seem inexpensive, bandwidth provided as a network service is not likely to become so cheap that wasting it will be the most cost-effective design principle. Even if low-cost bandwidth does eventually become commonly available, we do not accept that it will be available *everywhere* in the Internet. Unless we provide for the possibility of dealing with congested links, then real-time services will simply be precluded in those cases. We find that restriction unacceptable.

- "Simple priority is sufficient."

  It is true that simply giving higher priority to real-time traffic would lead to adequate real-time service at some times and under some conditions. But priority is an implementation mechanism, not a service model. If we define the service by means of a specific mechanism, we may not get the exact features we want. In the case of simple priority, the issue is that as soon as there are too many real-time streams competing for the higher priority, every stream is degraded. Restricting our service to this single failure mode is unacceptable. In some cases, users will demand that some streams succeed while some new requests receive a “busy signal”.

- “Applications can adapt.”

  The development of adaptive real-time applications, such as Jacobson’s audio program VAT, does not eliminate the need to bound packet delivery time. Human requirements for interaction and intelligibility limit the possible range of adaptation to network delays. We have seen in real experiments that, while VAT can adapt to network delays of many seconds, the users find that interaction is impossible in these cases.

We conclude that there is an inescapable requirement for routers to be able to reserve resources, in order to provide special QoS for specific user packet streams, or *flows*. This in turn requires flow-specific state in the routers, which represents an important and fundamental change to the Internet model. The Internet architecture was been founded on the concept that all flow-related state should be in the end systems [Clark88]. Designing the TCP/IP protocol suite on this concept led to a robustness that is one of the keys to its success. In section 5 we discuss how the flow state added to the routers for resource reservation can be made “soft”, to preserve the robustness of the Internet protocol suite.

There is a real-world side effect of resource reservation in routers. Since it implies that some users are getting privileged service, resource reservation will need enforcement of policy and administrative controls. This in turn will lead to two kinds of authentication requirements: authentication of users who make reservation requests, and authentication of packets that use the reserved resources. However, these issues are not unique to *IS*; other aspects of the evolution of the Internet, including commercialization and commercial security, are leading to the same requirements. We do not discuss the issues of policy or security further in this memo, but they will require attention.

We make another fundamental assumption, that it is desirable to use the Internet as a common infrastructure to support both non-real-time and real-time communication. One could alternatively build an entirely new, parallel infrastructure for real-time services, leaving the Internet unchanged. We reject this approach, as it would lose the significant advantages of statistical sharing between real-time and non-real-time traffic, and it would be much more complex to build and administer than a common infrastructure.

In addition to this assumption of common infrastructure, we adopt a unified protocol stack model, employing a single internet-layer protocol for both real-time and non-real-time service. Thus, we propose to use the existing internet-layer protocol (e.g., IP or CLNP) for real-time data. Another approach would be to add a new real-time protocol in the internet layer [ST2-90]. Our unified stack approach provides economy of mechanism, and it allows us to fold controlled link-sharing in easily. It also handles the problem of partial coverage, i.e., allowing interoperation between *IS*-capable Internet systems and systems that have not been extended, without the complexity of tunneling.

We take the view that there should be a single service model for the Internet. If there were different service models in different parts of the Internet, it is very difficult to see how any end-to-end service quality statements could be made. However, a single service model does not necessarily imply a single implementation for packet scheduling or admission control. Although specific packet scheduling and admission control mechanisms that satisfy our service model have been developed, it is quite possible that other mechanisms will also satisfy the service model. The reference implementation framework, introduced below, is intended to allow discussion of implementation issues without mandating a single design.

Based upon these considerations, we believe that an *IS* extension that includes additional flow state in routers and an explicit setup mechanism is necessary to provide the needed service. A partial solution short of this point would not be a wise investment. We believe that the extensions we propose preserve the essential robustness and efficiency of the Internet architecture, and they allow efficient management of the network resources; these will be important goals even if bandwidth becomes very inexpensive.

2.2 Reference Implementation Framework

We propose a reference implementation framework to realize the *IS* model. This framework includes four components: the *packet scheduler*, the *admission control* routine, the *classifier*, and the *reservation setup protocol*. These are discussed briefly below and more fully in Sections 4 and 5.

In the ensuing discussion, we define the "flow" abstraction as a distinguishable stream of related datagrams that results from a single user activity and requires the same QoS. For example, a flow might consist of one transport connection or one video stream between a given host pair. It is the finest granularity of packet stream distinguishable by the *IS*. We define a flow to be simplex, i.e., to have a single source but N destinations. Thus, an N-way teleconference will generally require N flows, one originating at each site.

In today's Internet, IP forwarding is completely egalitarian; all packets receive the same quality of service, and packets are typically forwarded using a strict FIFO queueing discipline. For integrated services, a router must implement an appropriate QoS for each flow, in accordance with the service model. The router function that creates different qualities of service is called "traffic control". Traffic control in turn is implemented by three components: the packet scheduler, the classifier, and admission control.

- Packet Scheduler

  The packet scheduler manages the forwarding of different packet streams using a set of queues and perhaps other mechanisms like timers. The packet scheduler must be implemented at the point where packets are queued; this is the output driver level of a typical operating system, and corresponds to the link layer protocol. The details of the scheduling algorithm may be specific to the particular output medium. For example, the output driver will need to invoke the appropriate link-layer controls when interfacing to a network technology that has an internal bandwidth allocation mechanism.

An experimental packet scheduler has been built that implements the *IS* model described in Section 3 and [SCZ93]; this is known as the CSZ scheduler and is discussed further in Section 4. We note that the CSZ scheme is not mandatory to accomplish our service model; indeed for parts of the network that are known always to be underloaded, FIFO will deliver satisfactory service.

There is another component that could be considered part of the packet scheduler or separate: the *estimator* [Jacobson91]. This algorithm is used to measure properties of the outgoing traffic stream, to develop statistics that control packet scheduling and admission control. This memo will consider the estimator to be a part of the packet scheduler.

- Classifier

  For the purpose of traffic control (and accounting), each incoming packet must be mapped into some *class*; all packets in the same class get the same treatment from the packet scheduler. This mapping is performed by the classifier. Choice of a class may be based upon the contents of the existing packet header(s) and/or some additional classification number added to each packet.

  A class might correspond to a broad category of flows, e.g., all video flows or all flows attributable to a particular organization. On the other hand, a class might hold only a single flow. A class is an abstraction that may be local to a particular router; the same packet may be classified differently by different routers along the path. For example, backbone routers may choose to map many flows into a few aggregated classes, while routers nearer the periphery, where there is much less aggregation, may use a separate class for each flow.

- Admission Control

  Admission control implements the decision algorithm that a router or host uses to determine whether a new flow can be granted the requested QoS without impacting earlier guarantees. Admission control is invoked at each node to make a local accept/reject decision, at the time a host requests a real-time service along some path through the Internet. The admission control algorithm must be consistent with the service model, and it is logically part of traffic control. Although there are still open research issues in admission control, a first cut exists [JCSZ92].

  Admission control is sometimes confused with *policing* or *enforcement*, which is a packet-by-packet function at the "edge" of the network to ensure that a host does not violate its promised traffic characteristics. We consider policing to be one of the functions of the packet scheduler.

  In addition to ensuring that QoS guarantees are met, admission control will be concerned with enforcing administrative policies on resource reservations. Some policies will demand authentication of those requesting reservations. Finally, admission control will play an important role in accounting and administrative reporting.

The fourth and final component of our implementation framework is a reservation setup protocol, which is necessary to create and maintain flow-specific state in the endpoint hosts and in routers along the path of a flow. Section 5 discusses a reservation setup protocol called RSVP (for "ReSerVation Protocol") [RSVP93a, RSVP93b]. It may not be possible to insist that there be only one reservation protocol in the Internet, but we will argue that multiple choices for reservation protocols will cause confusion. We believe that multiple protocols should exist only if they support different modes of reservation.

The setup requirements for the link-sharing portion of the service model are far less clear than those for resource reservations. While we expect that much of this can be done through network management interfaces, and thus need not be part of the overall architecture, we may also need RSVP to play a role in providing the required state.

In order to state its resource requirements, an application must specify the desired QoS using a list of parameters that is called a *flowspec* [Partridge92]. The flowspec is carried by the reservation setup protocol, passed to admission control for to test for acceptability, and ultimately used to parametrize the packet scheduling mechanism.

Figure 1 shows how these components might fit into an IP router that has been extended to provide integrated services. The router has two broad functional divisions: the forwarding path below the double horizontal line, and the background code above the line.

The forwarding path of the router is executed for every packet and must therefore be highly optimized. Indeed, in most commercial routers, its implementation involves a hardware assist. The forwarding path is divided into three sections: input driver, internet forwarder, and output driver. The internet forwarder interprets the internetworking protocol header appropriate to the protocol suite, e.g., the IP header for TCP/IP, or the CLNP header for OSI. For each packet, an internet forwarder executes a suite-dependent classifier and then passes the packet and its class to the appropriate output driver. A classifier must be both general and efficient. For efficiency, a common mechanism should be used for both resource classification and route lookup.

The output driver implements the packet scheduler. (Layerists will observe that the output driver now has two distinct sections: the packet scheduler that is largely independent of the detailed mechanics of the interface, and the actual I/O driver that is only concerned with the grittiness of the hardware. The estimator lives somewhere in between. We only note this fact, without suggesting that it be elevated to a principle.).

The background code is simply loaded into router memory and executed by a general-purpose CPU.

```
 -------------------------------------------------------------
|         ------------     ------------     -----------       |
|        |            |   | Reservation|   |           |      |
|        |   Routing  |   |    Setup   |   | Management|      |
|        |    Agent   |   |    Agent   |   |  Agent    |      |
|        |______._____|   |______._____|   |_____._____|      |
|               .                .    |          .            |
|               .                .   _V________  .            |
|               .                .  | Admission| .            |
|               .                .  |  Control | .            |
|               V                .  |__________| .            |
|          [Routing ]            V               V            |
|          [Database]      [Traffic Control Database]         |
|=============================================================|
|        |                 |     _______                      |
|        |   __________    |    |_|_|_|_| => o                |
|        |  |          |   |      Packet   |     _____        |
|    ====> |Classifier| =====>  Scheduler  |===>|_|_|_| ===>
|        |  |__________|   |     _______   |                  |
|        |                 |    |_|_|_|_| => o                |
| Input  |   Internet      |                                  |
| Driver |   Forwarder     |     O u t p u t   D r i v e r    |
|________|_________________|__________________________________|
```

Figure 1: Implementation Reference Model for Routers

These background routines create data structures that control the forwarding path. The routing agent implements a particular routing protocol and builds a routing database. The reservation setup agent implements the protocol used to set up resource reservations; see Section 5. If admission control gives the "OK" for a new request, the appropriate changes are made to the classifier and packet scheduler database to implement the desired QoS. Finally, every router supports an agent for network management. This agent must be able to modify the classifier and packet scheduler databases to set up controlled link-sharing and to set admission control policies.

The implementation framework for a host is generally similar to that for a router, with the addition of applications. Rather than being forwarded, host data originates and terminates in an application. An application needing a real-time QoS for a flow must somehow invoke a local reservation setup agent. The best way to interface to applications is still to be determined. For example, there might be an explicit API for network resource setup, or the setup might be invoked implicitly as part of the operating system scheduling function. The IP output routine of a host may need no classifier, since the class assignment for a packet can be specified in the local I/O control structure corresponding to the flow.

In routers, integrated service will require changes to both the forwarding path and the background functions. The forwarding path, which may depend upon hardware acceleration for performance, will be the more difficult and costly to change. It will be vital to choose a set of traffic control mechanisms that is general and adaptable to a wide variety of policy requirements and future circumstances, and that can be implemented efficiently.

3 Integrated Services Model

A service model is embedded within the network service interface invoked by applications to define the set of services they can request. While both the underlying network technology and the overlying suite of applications will evolve, the need for compatibility requires that this service interface remain relatively stable (or, more properly, extensible; we do expect to add new services in the future but we also expect that it will be hard to change existing services). Because of its enduring impact, the service model should not be designed in reference to any specific network artifact but rather should be based on fundamental service requirements.

We now briefly describe a proposal for a core set of services for the Internet; this proposed core service model is more fully described in [SCZ93a, SCZ93b]. This core service model addresses those services which relate most directly to the time-of-delivery of packets. We leave the remaining services (such as routing, security, or stream synchronization) for other standardization venues. A service model consists of a set of service commitments; in response to a service request the network commits to deliver some service. These service commitments can be categorized by the entity to whom they are made: they can be made to either individual flows or to collective entities (classes of flows). The service commitments made to individual flows are intended to provide reasonable application performance, and thus are driven by the ergonomic requirements of the applications; these service commitments relate to the quality of service delivered to an individual flow. The service commitments made to collective entities are driven by resource-sharing, or economic, requirements; these service commitments relate to the aggregate resources made available to the various entities.

In this section we start by exploring the service requirements of individual flows and propose a corresponding set of services. We then discuss the service requirements and services for resource sharing. Finally, we conclude with some remarks about packet dropping.

3.1 Quality of Service Requirements

The core service model is concerned almost exclusively with the time-of-delivery of packets. Thus, per-packet delay is the central quantity about which the network makes quality of service commitments. We make the even more restrictive assumption that the only quantity about which we make quantitative service commitments are bounds on the maximum and minimum delays.

The degree to which application performance depends on low delay service varies widely, and we can make several qualitative distinctions between applications based on the degree of their dependence. One class of applications needs the data in each packet by a certain time and, if the data has not arrived by then, the data is essentially worthless; we call these real-time applications. Another class of applications will always wait for data to arrive; we call these *elastic* applications. We now consider the delay requirements of these two classes separately.

3.1.1 Real-Time Applications

An important class of such real-time applications, which are the only real-time applications we explicitly consider in the arguments that follow, are *playback* applications. In a playback application, the source takes some signal, packetizes it, and then transmits the packets over the network. The network inevitably introduces some variation in the delay of the delivered packets. The receiver depacketizes the data and then attempts to faithfully play back the signal. This is done by buffering the incoming data and then replaying the signal at some fixed offset delay from the original departure time; the term *playback point* refers to the point in time which is offset from the original departure time by this fixed delay. Any data that arrives before its associated playback point can be used to reconstruct the signal; data arriving after the playback point is essentially useless in reconstructing the real-time signal.

In order to choose a reasonable value for the offset delay, an application needs some *a priori* characterization of the maximum delay its packets will experience. This *a priori* characterization could either be provided by the network in a quantitative service commitment to a delay bound, or through the observation of the delays experienced by the previously arrived packets; the application needs to know what delays to expect, but this expectation need not be constant for the entire duration of the flow.

The performance of a playback application is measured along two dimensions: latency and fidelity. Some playback applications, in particular those that involve interaction between the two ends of a connection such as a phone call, are rather sensitive to the latency; other playback applications, such as transmitting a movie or lecture, are not. Similarly, applications exhibit a wide range of sensitivity to loss of fidelity. We will consider two somewhat artificially dichotomous classes: intolerant applications, which require an absolutely faithful playback, and tolerant applications, which can tolerate some loss of fidelity. We expect that the vast bulk of audio and video applications will be tolerant, but we also suspect that there will be other applications, such as circuit emulation, that are intolerant.

Delay can affect the performance of playback applications in two ways. First, the value of the offset delay, which is determined by predictions about the future packet delays, determines the latency of the application. Second, the delays of individual packets can decrease the fidelity of the playback by exceeding the offset delay; the application then can either change the offset delay in order to play back late packets (which introduces distortion) or merely discard late packets (which creates an incomplete signal). The two different ways of coping with late packets offer a choice between an incomplete signal and a distorted one, and the optimal choice will depend on the details of the application, but the important point is that late packets necessarily decrease fidelity.

Intolerant applications must use a fixed offset delay, since any variation in the offset delay will introduce some distortion in the playback. For a given distribution of packet delays, this fixed offset delay must be larger than the absolute maximum delay, to avoid the possibility of late packets. Such an application can only set its offset delay appropriately if it is given a perfectly reliable upper bound on the maximum delay of each packet. We call a service characterized by a perfectly reliable upper bound on delay *guaranteed service*, and propose this as the appropriate service model for intolerant playback applications.

In contrast, tolerant applications need not set their offset delay greater than the absolute maximum delay, since they can tolerate some late packets. Moreover, instead of using a single fixed value for the offset delay, they can attempt to reduce their latency by varying their offset delays in response to the actual packet delays experienced in the recent past. We call applications which vary their offset delays in this manner *adaptive* playback applications.

For tolerant applications we propose a service model called *predictive service* which supplies a fairly reliable, but not perfectly reliable, delay bound. This bound, in contrast to the bound in the guaranteed service, is not based on worst case assumptions on the behavior of other flows. Instead, this bound might be computed with properly conservative predictions about the behavior of other flows. If the network turns out to be wrong and the bound is violated, the application's performance will perhaps suffer, but the users are willing to tolerate such interruptions in service in return for the presumed lower cost of the service. Furthermore, because many of the tolerant applications are adaptive, we augment the predictive service to also give *minimax* service, which is to attempt to minimize the ex post maximum delay. This service is not trying to minimize the delay of every packet, but rather is trying to pull in the tail of the delay distribution.

It is clear that given a choice, with all other things being equal, an application would perform no worse with absolutely reliable bounds than with fairly reliable bounds. Why, then, do we offer predictive service? The key consideration here is efficiency; when one relaxes the service requirements from perfectly to fairly reliable bounds, this increases the level of network utilization that can be sustained, and thus the price of the predictive service will presumably be lower than that of guaranteed service. The predictive service class is motivated by the conjecture that the performance penalty will be small for tolerant applications but the overall efficiency gain will be quite large.

In order to provide a delay bound, the nature of the traffic from the source must be characterized, and there must be some admission control algorithm which insures that a requested flow can actually be accommodated. A fundamental point of our overall architecture is that traffic characterization and admission control are necessary for these real-time delay bound services. So far we have assumed that an application's data generation process is an intrinsic property unaffected by the network. However, there are likely to be many audio and video applications which can adjust their coding scheme and thus can alter the resulting data generation process depending on the network service available. This alteration of the coding scheme will present a tradeoff between fidelity (of the coding scheme itself, not of the playback process) and the bandwidth requirements of the flow. Such *rate-adaptive* playback applications have the advantage that they can adjust to the current network conditions not just by resetting their playback point but also by adjusting the traffic pattern itself. For rate-adaptive applications, the traffic characterizations used in the service commitment are not immutable. We can thus augment the service model by allowing the network to notify (either implicitly through packet drops or explicitly through control packets) rate-adaptive applications to change their traffic characterization.

3.1.2 Elastic Applications

While real-time applications do not wait for late data to arrive, elastic applications will always wait for data to arrive. It is not that these applications are insensitive to delay; to the contrary, significantly increasing the delay of a packet will often harm the application's performance. Rather, the key point is that the application typically uses the arriving data immediately, rather than buffering it for some later time, and will always choose to wait for the incoming data rather than proceed without it. Because arriving data can be used immediately, these applications do not require any a priori characterization of the service in order for the application to function. Generally speaking, it is likely that for a given distribution of packet delays, the perceived performance of elastic applications will depend more on the average delay than on the tail of the delay distribution. One can think of several categories of such elastic applications: interactive burst (Telnet, X, NFS), interactive bulk transfer (FTP), and asynchronous bulk transfer (electronic mail, FAX). The delay requirements of these elastic applications vary from rather demanding for interactive burst applications to rather lax for asynchronous bulk transfer, with interactive bulk transfer being intermediate between them.

An appropriate service model for elastic applications is to provide *as-soon-as-possible*, or ASAP service. (For compatibility with historical usage, we will use the term best-effort service when referring to ASAP service.). We furthermore propose to offer several classes of best-effort service to reflect the relative delay sensitivities of different elastic applications. This service model allows interactive burst applications to have lower delays than interactive bulk applications, which in turn would have lower delays than asynchronous bulk applications. In contrast to the real-time service models, applications using this service are not subject to admission control.

The taxonomy of applications into tolerant playback, intolerant playback, and elastic is neither exact nor complete, but was only used to guide the development of the core service model. The resulting core service model should be judged not on the validity of the underlying taxonomy but rather on its ability to adequately meet the needs of the entire spectrum of applications. In particular, not all real-time applications are playback applications; for example, one might imagine a visualization application which merely displayed the image encoded in each packet whenever it arrived. However, non-playback applications can still use either the guaranteed or predictive real-time service model, although these services are not specifically tailored to their needs. Similarly, playback applications cannot be neatly classified as either tolerant or intolerant, but rather fall along a continuum; offering both guaranteed and predictive service allows applications to make their own tradeoff between fidelity, latency, and cost. Despite these obvious deficiencies in the taxonomy, we expect that it describes the service requirements of current and future applications well enough so that our core service model can adequately meet all application needs.

3.2 Resource-Sharing Requirements and Service Models

The last section considered quality of service commitments; these commitments dictate how the network must allocate its resources among the individual flows. This allocation of resources is typically negotiated on a flow-by-flow basis as each flow requests admission to the network, and does not address any of the policy issues that arise when one looks at collections of flows. To address these collective policy issues, we now discuss resource-sharing service commitments. Recall that for individual quality of service commitments we focused on delay as the only quantity of interest. Here, we postulate that the quantity of primary interest in resource-sharing is aggregate bandwidth on individual links. Thus, this component of the service model, called *link-sharing*, addresses the question of how to share the aggregate bandwidth of a link among various collective entities according to some set of specified shares. There are several examples that are commonly used to explain the requirement of link-sharing among collective entities.

Multi-entity link-sharing. – A link may be purchased and used jointly by several organizations, government agencies or the like. They may wish to insure that under overload the link is shared in a controlled way, perhaps in proportion to the capital investment of each entity. At the same time, they might wish that when the link is underloaded, any one of the entities could utilize all the idle bandwidth.

Multi-protocol link-sharing – In a multi-protocol Internet, it may be desired to prevent one protocol family (DECnet, IP, IPX, OSI, SNA, etc.) from overloading the link and excluding the other families. This is important because different families may have different methods of detecting and responding to congestion, and some methods may be more "aggressive" than others. This could lead to a situation in which one protocol backs off more rapidly than another under congestion, and ends up getting no bandwidth. Explicit control in the router may be required to correct this. Again, one might expect that this control should apply only under overload, while permitting an idle link to be used in any proportion.

Multi-service sharing – Within a protocol family such as IP, an administrator might wish to limit the fraction of bandwidth allocated to various service classes. For example, an administrator might wish to limit the amount of real-time traffic to some fraction of the link, to avoid preempting elastic traffic such as FTP.

In general terms, the link-sharing service model is to share the aggregate bandwidth according to some specified shares. We can extend this link-sharing service model to a hierarchical version. For instance, a link could be divided between a number of organizations, each of which would divide the resulting allocation among a number of protocols, each of which would be divided among a number of services. Here, the sharing is defined by a tree with shares assigned to each leaf node.

An idealized fluid model of instantaneous link-sharing with proportional sharing of excess is the fluid processor sharing model (introduced in [DKS89] and further explored in [Parekh92] and generalized to the hierarchical case) where at every instant the available bandwidth is shared between the active entities (i.e., those having packets in the queue) in proportion to the assigned shares of the resource. This fluid model exhibits the desired policy behavior but is, of course, an unrealistic idealization. We then propose that the actual service model should be to approximate, as closely as possible, the bandwidth shares produced by this ideal fluid model. It is not necessary to require that the specific order of packet departures match those of the fluid model since we presume that all detailed per-packet delay requirements of individual flows are addressed through quality of service commitments and, furthermore, the satisfaction with the link-sharing service delivered will probably not depend very sensitively on small deviations from the scheduling implied by the fluid link-sharing model.

We previously observed that admission control was necessary to ensure that the real-time service commitments could be met. Similarly, admission control will again be necessary to ensure that the link-sharing commitments can be met. For each entity, admission control must keep the cumulative guaranteed and predictive traffic from exceeding the assigned link-share.

3.3 Packet Dropping

So far, we have implicitly assumed that all packets within a flow were equally important. However, in many audio and video streams, some packets are more valuable than others. We therefore propose augmenting the service model with a *preemptable* packet service, whereby some of the packets within a flow could be marked as preemptable. When the network was in danger of not meeting some of its quantitative service commitments, it could exercise a certain packet's "preemptability option" and discard the packet (not merely delay it, since that would introduce out-of-order problems). By discarding these preemptable packets, a router can reduce the delays of the not-preempted packets.

Furthermore, one can define a class of packets that is not subject to admission control. In the scenario described above where preemptable packets are dropped only when quantitative service commitments are in danger of being violated, the expectation is that preemptable packets will almost always be delivered and thus they must included in the traffic description used in admission control. However, we can extend preemptability to the extreme case of *expendable* packets (the term expendable is used to connote an extreme degree of preemptability), where the expectation is that many of these expendable packets may not be delivered. One can then exclude expendable packets from the traffic description used in admission control; i.e., the packets are not considered part of the flow from the perspective of admission control, since there is no commitment that they will be delivered.

3.4 Usage Feedback

Another important issue in the service is the model for usage feedback, also known as "accounting", to prevent abuse of network resources. The link-sharing service described earlier can be used to provide administratively-imposed limits on usage. However, a more free-market model of network access will require back-pressure on users for the network resources they reserve. This is a highly contentious issue, and we are not prepared to say more about it at this time.

3.5 Reservation Model

The *reservation model* describes how an application negotiates for a QoS level. The simplest model is that the application asks for a particular QoS and the network either grants it or refuses. Often the situation will be more complex. Many applications will be able to get acceptable service from a range of QoS levels, or more generally, from anywhere within some region of the multi-dimensional space of a flowspec.

For example, rather than simply refusing the request, the network might grant a lower resource level and inform the application of what QoS has been actually granted. A more complex example is the *two-pass* reservation model, In this scheme, an "offered" flowspec is propagated along the multicast distribution tree from each sender Si to all receivers Rj. Each router along the path records these values and perhaps adjusts them to reflect available capacity. The receivers get these offers, generate corresponding "requested" flowspecs, and propagate them back along the same routes to the senders. At each node, a local reconciliation must be performed between the offered and the requested flowspec to create a reservation, and an appropriately modified requested flowspec is passed on. This two-pass scheme allows extensive properties like allowed delay to be distributed across hops in the path [Tenet90, ST2-90]. Further work is needed to define the amount of generality, with a corresponding level of complexity, that is required in the reservation model.

4 Traffic Control Mechanisms

We first survey very briefly the possible traffic control mechanisms. Then in Section 4.2 we apply a subset of these mechanisms to support the various services that we have proposed.

4.1 Basic Functions

In the packet forwarding path, there is actually a very limited set of actions that a router can take. Given a particular packet, a router must select a route for it; in addition the router can either forward it or drop it, and the router may reorder it with respect to other packets waiting to depart. The router can also hold the packet, even though the link is idle. These are the building blocks from which we must fashion the desired behavior.

4.1.1 Packet Scheduling

The basic function of packet scheduling is to reorder the output queue. There are many papers that have been written on possible ways to manage the output queue, and the resulting behavior. Perhaps the simplest approach is a priority scheme, in which packets are ordered by priority, and highest priority packets always leave first. This has the effect of giving some packets absolute preference over others; if there are enough of the higher priority packets, the lower priority class can be completely prevented from being sent.

An alternative scheduling scheme is round-robin or some variant, which gives different classes of packets access to a share of the link. A variant called Weighted Fair Queueing, or WFQ, has been demonstrated to allocate the total bandwidth of a link into specified shares.

There are more complex schemes for queue management, most of which involve observing the service objectives of individual packets, such as delivery deadline, and ordering packets based on these criteria.

4.1.2 Packet Dropping

The controlled dropping of packets is as important as their scheduling.

Most obviously, a router must drop packets when its buffers are all full. This fact, however, does not determine which packet should be dropped. Dropping the arriving packet, while simple, may cause undesired behavior.

In the context of today's Internet, with TCP operating over best effort IP service, dropping a packet is taken by TCP as a signal of congestion and causes it to reduce its load on the network. Thus, picking a packet to drop is the same as picking a source to throttle. Without going into any particular algorithm, this simple relation suggests that some specific dropping controls should be implemented in routers to improve congestion control.

In the context of real-time services, dropping more directly relates to achieving the desired quality of service. If a queue builds up, dropping one packet reduces the delay of all the packets behind it in the queue. The loss of one can contribute to the success of many. The problem for the implementor is to determine when the service objective (the delay bound) is in danger of being violated. One cannot look at queue length as an indication of how long packets have sat in a queue. If there is a priority scheme in place, packets of lower priority can be pre-empted indefinitely, so even a short queue may have very old packets in it. While actual time stamps could be used to measure holding time, the complexity may be unacceptable.

Some simple dropping schemes, such as combining all the buffers in a single global pool, and dropping the arriving packet if the pool is full, can defeat the service objective of a WFQ scheduling scheme. Thus, dropping and scheduling must be coordinated.

4.1.3 Packet Classification

The above discussion of scheduling and dropping presumed that the packet had been classified into some flow or sequence of packets that should be treated in a specified way. A preliminary to this sort of processing is the classification itself. Today a router looks at the destination address and selects a route. The destination address is not sufficient to select the class of service a packet must receive; more information is needed.

One approach would be to abandon the IP datagram model for a virtual circuit model, in which a circuit is set up with specific service attributes, and the packet carries a circuit identifier. This is the approach of ATM as well as protocols such as ST-II [ST2-90]. Another model, less hostile to IP, is to allow the classifier to look at more fields in the packet, such as the source address, the protocol number and the port fields. Thus, video streams might be recognized by a particular well-known port field in the UDP header, or a particular flow might be recognized by looking at both the source and destination port numbers. It would be possible to look even deeper into the packets, for example testing a field in the application layer to select a subset of a hierarchically-encoded video stream.

The classifier implementation issues are complexity and processing overhead. Current experience suggests that careful implementation of efficient algorithms can lead to efficient classification of IP packets. This result is very important, since it allows us to add QoS support to existing applications, such as Telnet, which are based on existing IP headers.

One approach to reducing the overhead of classification would be to provide a *flow-id* field in the Internet-layer packet header. This flow-id would be a handle that could be cached and used to short-cut classification of the packet. There are a number of variations of this concept, and engineering is required to choose the best design.

4.1.4 Admission Control

As we stated in the introduction, real-time service depends on setting up state in the router and making commitments to certain classes of packets. In order to insure that these commitments can be met, it is necessary that resources be explicitly requested, so that the request can be refused if the resources are not available. The decision about resource availability is called admission control.

Admission control requires that the router understand the demands that are currently being made on its assets. The approach traditionally proposed is to remember the service parameters of past requests, and make a computation based on the worst-case bounds on each service. A recent proposal, which is likely to provide better link utilization, is to program the router to measure the actual usage by existing packet flows, and to use this measured information as a basis of admitting new flows [JCSZ92]. This approach is subject to higher risk of overload, but may prove much more effective in using bandwidth.

Note that while the need for admission control is part of the global service model, the details of the algorithm run in each router is a local matter. Thus, vendors can compete by developing and marketing better admission control algorithms, which lead to higher link loadings with fewer service overloads.

4.2 Applying the Mechanisms

The various tools described above can be combined to support the services which were discussed in section 3.

- Guaranteed delay bounds

  A theoretical result by Parekh [Parekh92] shows that if the router implements a WFQ scheduling discipline, and if the nature of the traffic source can be characterized (e.g. if it fits within some bound such as a token bucket) then there will be an absolute upper bound on the network delay of the traffic in question. This simple and very powerful result applies not just to one switch, but to general networks of routers. The result is a constructive one; that is, Parekh displays a source behavior which leads to the bound, and then shows that this behavior is the worst possible. This means that the bound he computes is the best there can be, under these assumptions.

- Link sharing

  The same WFQ scheme can provide controlled link sharing. The service objective here is not to bound delay, but to limit overload shares on a link, while allowing any mix of traffic to proceed if there is spare capacity. This use of WFQ is available in commercial routers today, and is used to segregate traffic into classes based on such things as protocol type or application. For example, one can allocate separate shares to TCP, IPX and SNA, and one can assure that network control traffic gets a guaranteed share of the link.

- Predictive real-time service

  This service is actually more subtle than guaranteed service. Its objective is to give a delay bound which is, on the one hand, as low as possible, and on the other hand, stable enough that the receiver can estimate it. The WFQ mechanism leads to a guaranteed bound, but not necessarily a low bound. In fact, mixing traffic into one queue, rather than separating it as in WFQ, leads to lower bounds, so long as the mixed traffic is generally similar (e.g., mixing traffic from multiple video coders makes sense, mixing video and FTP does not).

  This suggests that we need a two-tier mechanism, in which the first tier separates traffic which has different service objectives, and the second tier schedules traffic within each first tier class in order to meet its service objective.

4.3 An example: The CSZ scheme

As a proof of concept, a code package has been implemented which realizes the services discussed above. It actually uses a number of the basic tools, combined in a way specific to the service needs. We describe in general terms how it works, to suggest how services can be realized. We stress that there are other ways of building a router to meet the same service needs, and there are in fact other implementations being used today.

At the top level, the CSZ code uses WFQ as an isolation mechanism to separate guaranteed flows from each other, as well as from the rest of the traffic. Guaranteed service gets the highest priority when and only when it needs the access to meets its deadline. WFQ provides a separate guarantee for each and every guaranteed flow.

Predictive service and best effort service are separated by priority. Within the predictive service class, a further priority is used to provide sub-classes with different delay bounds. Inside each predictive sub-class, simple FIFO queueing is used to mix the traffic, which seems to produce good overall delay behavior. This works because the top-tier algorithm has separated out the best effort traffic such as FTP.

Within the best-effort class, WFQ is used to provide link sharing. Since there is a possible requirement for nested shares, this WFQ code can be used recursively. There are thus two different uses of WFQ in this code, one to segregate the guaranteed classes, and one to segregate the link shares. They are similar, but differ in detail.

Within each link share of the best effort class, priority is used to permit more time-sensitive elastic traffic to precede other elastic traffic, e.g., to allow interactive traffic to precede asynchronous bulk transfers.

The CSZ code thus uses both WFQ and priority in an alternating manner to build a mechanism to support a range of rather sophisticated service offerings. This discussion is very brief, and does not touch on a number of significant issues, such as how the CSZ code fits real time traffic into the link sharing objectives. But the basic building blocks are very simple, and very powerful. In particular, while priority has been proposed as a key to real-time services, WFQ may be the more general and powerful of the two schemes. It, rather than priority, supports guaranteed service and link sharing.

5 Reservation Setup Protocol

There are a number of requirements to be met by the design of a reservation setuop protocol. It should be fundamentally designed for a multicast environment, and it must accommodate hetero-

```
     Senders                                  Receivers

                   ---------------------
                  (                     ) ===> R1
        S1 ===>   (    Multicast        )
                  (                     ) ===> R2
                  (    distribution     )
        S2 ===>   (                     )
                  (                     ) ===> R3
                  (_____________________)
```

Figure 2: Multicast Distribution Session

geneous service needs. It must give flexible control over the manner in which reservations can be shared along branches of the multicast delivery trees. It should be designed around the elementary action of adding one sender and/or receiver to an existing set, or deleting one. It must be robust and scale well to large multicast groups. Finally, it must provide for advance reservation of resources, and for the preemption that this implies. The reservation setup protocol RSVP has been designed to meet these requirements [RSVP93a, RSVP93b]. This section gives an overview of the design of RSVP.

5.1 RSVP Overview

Figure 2 shows multi-source, multi-destination data delivery for a particular shared, distributed application. The arrows indicate data flow from senders S1 and S2 to receivers R1, R2, and R3, and the cloud represents the distribution mesh created by the multicast routing protocol. Multicasting distribution replicates each data packet from a sender Si, for delivery to every receiver Rj. We treat uncast delivery from S1 to R1 as a special case, and we call this multicast distribution mesh a *session*. A session is defined by the common IP (multicast) destination address of the receiver(s).

5.1.1 Flowspecs and Filter Specs

In general, an RSVP reservation request specifies the amount of resources to be reserved for all, or some subset of, the packets in a particular session. The resource quantity is specified by a *flowspec*, while the packet subset to receive those resources is specified by a *filter spec*. Assuming admission control succeeds, the flowspec will be used to parametrize a resource class in the packet scheduler, and the filter spec will be instantiated in the packet classifier to map the appropriate packets into this class. The subset of the classifier state that selects a particular class is referred to in RSVP documentation as a (packet) *filter*.

The RSVP protocol mechanisms provide a very general facility for creating and maintaining distributed reservation state across the mesh of multicast delivery paths. These mechanisms treat flowspecs and filter specs as mostly opaque binary data, handing them to the local traffic control machinery for interpretation. Of course, the service model presented to an application must specify how to encode flowspecs and filter specs.

5.1.2 Reservation Styles

RSVP offers several different reservation *styles*, which determine the manner in which the resource requirements of multiple receivers are aggregated in the routers. These styles allow the reserved resources to more efficiently meet application requirements. Currently there are three reservation styles, *wildcard*, *fixed-filter*, and *dynamic-filter*. A wildcard reservation uses a filter spec that is not source-specific, so all packets destined for the associated destination (session) may use a common pool of reserved resources. This allows a single resource allocation to be made across all distribution paths for the group. The wildcard reservation style is useful in support of an audio conference, where at most a small number of sources are active simultaneously and may share the resource allocation.

The other two styles use filter specs that select particular sources. A receiver may desire to receive from a fixed set of sources, or instead it may desire the network to switch between different source, by changing its filter spec(s) dymamically. A fixed-filter style reservation cannot be changed during its lifetime without re-invoking admission control. Dynamic-filter reservations do allow a receiver to modify its choice of source(s) over time without additional admission control; however, this requires that sufficient resources be allocated to handle the worst case when all downstream receivers take input from different sources.

5.1.3 Receiver Initiation

An important design question is whether senders or receivers should have responsibility for initiating reservations. A sender knows the qualities of the traffic stream it can send, while a receiver knows what it wants to (or can) receive. Perhaps the most obvious choice is to let the sender initiate the reservation. However, this scales poorly for large, dynamic multicast delivery trees and for heterogeneous receivers.

Both of these scaling problems are solved by making the receiver responsible for initiating a reservation. Receiver initiation handles heterogeneous receivers easily; each receiver simply asks for a reservation appropriate to itself, and any differences among reservations from different receivers are resolved ("merged") within the network by RSVP. Receiver initiation is also consisent with IP multicast, in which a multicast group is created implicitly by receivers joining it.

Although receiver-initiated reservation is the natural choice for multicast sessions, the justification for receiver initiateion may appear weaker for unicast sessions, where the sender may be the logical session initiator. However, we expect that every realtime application will have its higher-level signalling and control protocol, and this protocol can be used to signal the receiver to initiate a reservation (and perhaps indicate the flowspec to be used). For simplicity and economy, a setup protocol should support only one direction of initiation, and, and receiver initiation appears to us to be the clear winner.

RSVP uses receiver-initiation of rservations [RSVP93b]. A receiver is assumed to learn the senders' offered flowspecs by a higher-level mechanism ("out of band"), it then generates its own desired flowspec and propagates it towards the senders, making reservations in each router along the way.

5.1.4 Soft State

There are two different possible styles for reservation setup protocols, the "hard state" (HS) approach (also called "connection-oriented"), and the "soft state" (SS) approach (also called "connectionless"). In both approaches, multicast distribution is performed using flow-specific state in each router along the path. Under the HS approach, this state is created and deleted in a fully deterministic manner by cooperation among the routers. Once a host requests a session, the "network" takes responsibility for creating and later destroying the necessary state. ST-II is an example of the HS approach [ST2-90]. Since management of HS session state is completely deterministic, the HS setup protocol must be reliable, with acknowledgments and retransmissions. In order to achieve deterministic cleanup of state after a failure, there must be some mechanism to detect failures, i.e., an "up/down" protocol. The router upstream (towards the source) from a failure takes responsibility for rebuilding the necessary state on the router(s) along an alternate route.

RSVP takes the SS approach, which regards the reservation state as cached information that is installed and periodically refreshed by the end hosts. Unused state is timed out by the routers. If the route changes, the refresh messages automatically install the necessary state along the new route. The SS approach was chosen to obtain the simplicity and robustness that have been demonstrated by connectionless protocols such as IP [Clark88].

5.2 Routing and Reservations

There is a fundamental interaction between resource reservation set up and routing, since reservation requires the installation of flow state along the route of data packets. If and when a route changes, there must be some mechanism to set up a reservation along the new route.

Some have suggested that reservation setup necessarily requires route set up, i.e., the imposition of a virtual-circuit internet layer. However, our goal is to simply extend the Internet architecture, not replace it. The fundamental connectionless internet layer [Clark88] has been highly successful, and we wish to retain it as an architectural foundation. We propose instead to modify somewhat the pure datagram forwarding mechanism of the present Internet to accomodate *IS*.

There are four routing issues faced by a reservation setup protocol such as RSVP.

1. Find a route that supports resource reservation.

   This is simply "type-of-service" routing, a facility that is already available in some modern routing protocols.

2. Find a route that has sufficient unreserved capacity for a new flow.

   Early experiments on the ARPANET showed that it is difficult to do load-dependent dynamic routing on a packet-by-packet basis without instability problems. However, instability should not be a problem if load-dependent routing is performed only at reservation setup time.

   Two different approaches might be taken to finding a route with enough capacity. One could modify the routing protocol(s) and interface them to the traffic control mechanism, so the route computation can consider the average recent load. Alternatively, the routing protocol could be (re-)designed to provide multiple alternative routes, and reservation setup could be attempted along each in turn.

3. Adapt to a route failure

   When some node or link fails, adaptive routing finds an alternate path. The periodic refresh messages of RSVP will automatically request a reservation along the new path. Of course, this reservation may fail because there is insufficienct available capacity on the new path. This is a problem of provisioning and network engineering, which cannot be solved by the routing or setup protocols.

   There is a problem of timeliness of establishing reservation state on the new path. The end-to-end robustness mechanism of refreshes is limited in frequency by overhead, which may cause a gap in realtime service when an old route breaks and a new one is chosen. It should be possible to engineer RSVP to sypplement the global refresh mechanism with a local repair mechanism, using hints about route changes from the routing mechanism.

4. Adapt to a route change (without failure)

   Route changes may occur even without failure in the affected path. Although RSVP could use the same repair techniques as those described in (3), this case raises a problem with the robustness of the QoS guarantees. If it should happen that admission control fails on the new route, the user will see service degradation unnecessarily and capriciously, since the orginal route is still functional.

   To avoid this problem, a mechanism called "route pinning" has been suggested. This would modify the routing protocol implementation and the interface to the classifier, so that routes associated with resource reservations would be "pinned". The routing prootocol would not change a pinned route if it was still viable.

It may eventually be possible to fold together the routing and reservation setup problems, but we do not yet understand enough to do that. Furthermore, the reservation protocol needs to coexist with a number of different routing protocols in use in the Internet. Therefore, RSVP is currently designed to work with any current-generation routing protocol without modification. This is a short-term compromise, which may result in an occasional failure to create the best, or even any, real-time session, or an occasional service degradation due to a route change. We expect that future generations of routing protocols will remove this compromise, by including hooks and mechanisms that, in conjunction with RSVP, will solve the problems (1) through (4) just listed. They will support route pinning, notification of RSVP to trigger local repair, and selection of routes with *IS* support and adequate capacity.

The last routing-related issue is provided by mobile hosts. Our conjecture is that mobility is not essentially different from other route changes, so that the mechanism suggested in (3) and (4) will suffice. More study and experimentation is needed to prove or disprove this conjecture.

6 ACKNOWLEDGMENTS

Many Internet researchers have contributed to the work described in this memo. We want to especially acknowledge, Steve Casner, Steve Deering, Deborah Estrin, Sally Floyd, Shai Herzog, Van Jacobson, Sugih Jamin, Craig Partridge, John Wroclawski, and Lixia Zhang. This approach to Internet integrated services was initially discussed and organized in the End-to-End Research Group of the Internet Research Taskforce, and we are grateful to all members of that group for their interesting (and sometimes heated) discussions.

References

[CerfKahn74] Cerf, V., and R. Kahn, *A Protocol for Packet Network Intercommunication*, IEEE Trans on Comm., Vol. Com-22, No. 5, May 1974.

[Clark88] Clark, D., *The Design Philosophy of the DARPA Internet Protocols*, ACM SIGCOMM '88, August 1988.

[CSZ92] Clark, D., Shenker, S., and L. Zhang, *Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanisms*, Proc. SIGCOMM '92, Baltimore, MD, August 1992.

[DKS89] Demers, A., Keshav, S., and S. Shenker. *Analysis and Simulation of a Fair Queueing Algorithm*, Journal of Internetworking: Research and Experience, 1, pp. 3-26, 1990. Also in Proc. ACM SIGCOMM '89, pp 3-12.

[SCZ93a] Shenker, S., Clark, D., and L. Zhang, *A Scheduling Service Model and a Scheduling Architecture for an Integrated Services Packet Network*, submitted to ACM/IEEE Trans. on Networking.

[SCZ93b] Shenker, S., Clark, D., and L. Zhang, *A Service Model for the Integrated Services Internet*, Work in Progress, October 1993.

[Floyd92] Floyd, S., *Issues in Flexible Resource Management for Datagram Networks*, Proceedings of the 3rd Workshop on Very High Speed Networks, March 1992.

[Jacobson91] Jacobson, V., *Private Communication*, 1991.

[JCSZ92] Jamin, S., Shenker, S., Zhang, L., and D. Clark, *An Admission Control Algorithm for Predictive Real-Time Service*, Extended abstract, in Proc. Third International Workshop on Network and Operating System Support for Digital Audio and Video, San Diego, CA, Nov. 1992, pp. 73-91.

[Parekh92] Parekh, A., *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks*, Technical Report LIDS-TR-2089, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1992.

[Partridge92] Partridge, C., *A Proposed Flow Specification*, RFC 1363, BBN, July 1992.

[RSVP93a] Zhang, L., Deering, S., Estrin, D., Shenker, S., and D. Zappala, *RSVP: A New Resource ReSerVation Protocol*, Accepted for publication in IEEE Network, 1993.

[RSVP93b] Zhang, L., Braden, R., Estrin, D., Herzog, S., and S. Jamin, *Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification*, Work in Progress, 1993.

[ST2-90] Topolcic, C., *Experimental Internet Stream Protocol: Version 2 (ST-II)*, RFC 1190, BBN, October 1990.

[Tenet90] Ferrari, D., and D. Verma, *A Scheme for Real-Time Channel Establishment in Wide-Area Networks*, IEEE JSAC, Vol. 8, No. 3, pp 368-379, April 1990.

Security Considerations

As noted in Section 2.1, the ability to reserve resources will create a requirement for authentication, both of users requesting resource guarantees and of packets that claim to have the right to use those guarantees. These authentication issues are not otherwise addressed in this memo, but are for further study.

Authors' Addresses

Bob Braden
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292

Phone: (310) 822-1511
EMail: Braden@ISI.EDU

David Clark
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139-1986

Phone: (617) 253-6003
EMail: ddc@LCS.MIT.EDU

Scott Shenker
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

Phone: (415) 812-4840
EMail: Shenker@PARC.XEROX.COM