Internet Draft
Expires: April 1994
File: draft-shenker-realtime-model-00.ps

Scott Shenker
Xerox PARC
David D. Clark
MIT
Lixia Zhang
Xerox PARC
October 1993

A Service Model for an Integrated Services Internet

Status of Memo

Abstract

The Internet is currently being confronted with service demands from a new generation of applications. Supporting these applications effectively and efficiently will require extending the current Internet "best-effort" service model to one that offers an integrated suite of services. The purpose of this memo (which is derived primarily from [34]) is to describe a proposed "core" service model for an integrated services Internet. In the Appendix we discuss the process by which such a service model could be standardized by the IETF.

1 Introduction

The current Internet offers a very simple service model: all packets receive the same "best effort" service. The term "best effort" means that the network tries to forward packets as soon as possible, but makes no quantitative commitments about the quality of service delivered. This service model can be realized by using a single FIFO queue to do packet scheduling in the switches; in fact, this service model arose precisely because FIFO packet scheduling, without admission control, cannot efficiently deliver any other service model. This single class "best effort" service model provides the same quality of service to all flows[1]; this uniform quality of service is good, as measured by delay and dropped packets, when the network is lightly loaded but can be quite poor when the network is heavily utilized. Consequently, only those applications that are rather tolerant of this variable

[1] *Flow* is the term we use to refer to end-to-end connections and other more general varieties of traffic streams. We will return to this issue in Section 3.3 where we discuss multicast flows more explicitly.

service, such as file transfer (e.g., FTP), electronic mail, and interactive terminals (e.g., Telnet) have become widely adopted in the Internet.

However, we expect there to soon be widespread demand for an emerging generation of computer based applications, such as FAX, remote video, multimedia conferencing, data fusion, remote X-terminals, visualization, and virtual reality. These applications represent a wide variety of quality of service requirements, ranging from the asynchronous nature of FAX and electronic mail to the extremely time-sensitive nature of high quality audio, and from the low bandwidth requirements of Telnet to the bandwidth intensive requirements of HDTV. To meet all of these service requirements using the current Internet service model, it would be necessary (but perhaps not sufficient) to keep the utilization level extremely low. We think a better solution is to offer a more sophisticated service model, so that applications can specify their service needs and the network can then allocate its resources selectively towards those applications that are more performance sensitive. It is important to emphasize that this solution requires that applications explicitly specify their service desires; these needs are not derived implicitly by the network through the inspection of port numbers.

The service model is the enduring, and therefore the most fundamental, part of a network architecture. The service model will be incorporated into the network service interface used by future applications; as such, it will define the set of services they can request, and will therefore influence the design of future applications as well as the performance of existing ones. Thus, the service model should not be designed in reference to any specific network artifact but rather should be based on fundamental service requirements. 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. Actually, compatibility only demands that the existing parts of the service model must remain largely unchanged; the service model should be extensible with augmentations handled without difficulty. Also, we should note that these compatibility arguments apply *only* to those aspects of the service model which are part of the network service interface; the service model will also have some components (e.g., the link-sharing services as defined in Section 4) which are exercised through a network management interface, and here the compatibility arguments do not apply with nearly the same force.

This memo proposes a core service model for the Internet. We address those services which relate most directly to the time-of-delivery of packets. We do not address those services which are concerned with which network links are used (which is the domain of routing) and those services which involve encryption, security, authentication, or transmission reliability. We also do not consider services, such as reliable multicast, which do tangentially involve the time-of-delivery but which more fundamentally involve other factors such as buffer management and inter-switch acknowledgment algorithms. Furthermore, we do not consider time-of-delivery services which can best be delivered at the end host or by gateway switches at the edge of the network, such as synchronization of different traffic streams. Although many of the services listed above may perhaps be offered in the future, we do not expect that they will affect the basic core of the service model which we discuss here.

In order to efficiently support this more sophisticated service model, Internet routers must employ an appropriately sophisticated non-FIFO packet scheduling algorithm. In fact, the packet scheduling algorithm is the most fundamental way in which the network can allocate resources selectively; the network can also allocate selectively via routing or buffer management algorithms, but neither of these by themselves can support a sufficiently general service model (see [3, 10, 12, 14, 15, 23, 26, 28, 29, 35] for a few examples of packet scheduling algorithms). However, packet scheduling algorithms are only part of a complete mechanism to support explicit qualities of service. In particular, since resources are finite, one cannot always support an unbounded number of service requests. The network must employ some form of admission algorithm so that it has control over which service commitments are made. The admission process requires that flows characterize their traffic stream to the network when requesting service, and the network then determines whether or not to grant the service request. It is important to keep in mind that admission control plays a crucial role in allowing these scheduling algorithms to be effective by keeping the aggregate traffic load down to a level where meeting the service commitments is feasible (see [12, 17, 21, 25] for a few examples of admission control algorithms). In fact, admission control is but one kind of denial of service; we will discuss the several varieties of denial of service and their role in allowing the scheduling algorithm to meet service commitments.

This memo has 6 sections. In Section 2 we identify the two kinds of service commitments we expect future networks to make; these are quality of service commitments to individual flows and resource-sharing commitments to collective entities. In Section 3 we explore the service requirements of individual flows and then propose a corresponding set of service models. In Section 4 we discuss the service requirements for resource-sharing commitments to collective entities, and propose a related service model. In Section 5, we review the various forms denial of service can manifest, and the ways in which denial of service can be used to augment the core service model. We then conclude in Section 6 by discussing other viewpoints. In an Appendix we discuss how the Internet community can standardize a new service model.

2 Service Commitments

A service model is made up of service commitments; that is, a service model describes what service the network commits to deliver in response to a particular service request. In this section, we describe the various different kinds of service commitments that are included in our core service model.

Service commitments can be divided up into two classes, depending on the way in which the service is characterized. One class of service commitment is a quantitative or absolute service commitment, which is some form of assurance that the network service will meet or exceed the agreed upon quantitative specifications; a typical example of this is a bound on maximal packet delay. The other class of service commitment is a qualitative or relative service commitment, which is merely some form of assurance about how one set of packets will be treated relative to other sets of packets. One example of this kind of relative service commitment is to offer several different priority classes; the service in any priority class is not quantitatively characterized, but there is a relative commitment to serve traffic in a given priority class before traffic in lower priority classes. Thus, when we say that the current Internet offers only a single "best-effort" class of service, this is equivalent to saying that it does not offer any quantitative service commitments, and only offers the most trivial relative service commitment to treat all packets equivalently. An important distinction between these two classes of commitments is that quantitative service commitments often inherently require some form of admission control, with the flow characterizing its traffic in some manner; in contrast, relative service commitments generally do not require any admission control.

Service commitments can also be divided into two categories depending on the entities to which the commitments are made. The first category of service commitments is the one most often considered in the current literature; these are quality of service commitments to individual flows. In this case the network provides some form of assurance that the quality of service delivered to the contracting flow will meet or exceed the agreed upon specifications. The need for these kinds of service commitments is usually driven by the ergonomic requirements of individual applications. For instance, the perceived quality of many interactive audio and video applications declines dramatically when the delay of incoming packets becomes too large; thus, these applications would perform better if the network would commit to a small bound on the maximum packet queueing delay. In Section 3 we discuss what quality of service commitments are included in our core service model.

In contrast, the second category of service commitment we consider has rarely been explicitly discussed in the research literature, even though there is widespread agreement in the industry that there is great customer demand for this feature; these are resource-sharing commitments to collective entities. In this case, the network provides an assurance that the resource in question will be shared according to some prearranged convention among some set of collective entities. These collective entities could, for example, be institutions, protocol families, or application types. An example of the need for such resource-sharing commitments is when two private companies choose to jointly purchase a fiber optic link and then elect to share the bandwidth in proportion to the capital investments of the two companies. In Section 4, we present a more detailed motivation for this form of service commitment and then discuss the particular resource-sharing commitments that are part of our core service model.

We should reiterate that because the quality of service service commitments to individual flows will typically be invoked through the service interface, compatibility requires that their definition remain relatively stable. The resource sharing commitments will typically be invoked through a network management interface, not through the service interface used by applications, and therefore the need for compatibility does not require such a stable service definition.

3 Quality of Service Requirements and Service Models

In the previous section, we distinguished two sorts of service requirements, quality of service requirements and resource sharing requirements. In this section we consider quality of service requirements. We first argue that packet delay is the key measure of quality of service. We then present our assumptions about the nature of future computer-based applications and their service requirements. Finally, we describe a set of quality of service commitments designed to meet these service requirements.

3.1 The Centrality of Delay

There is one measure of service that is relevant to almost all applications: per-packet delay. In some sense, delay is the fundamental measure of the service given to a packet, since it describes when (and if) a packet is delivered and, if we assume that data is never corrupted (which we think is a good approximation for the future Internet[2]), the time of delivery is the only quantity of interest to applications. Delay is clearly the most central quality of service, and we will therefore start by assuming that the only qualities of service about which the network makes commitments relate to per-packet delay. Later, in Section 3.3 we will return to this point and ask if the service model that results from this initial assumption is sufficiently general.

In addition to restricting our attention to delay, 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. Thus, we have excluded quantitative service commitments about other delay related qualities of service, such as targets for average delay. This is based on three judgments. First, controlling nonextremal values of delay through packet scheduling algorithms is usually impractical because it requires detailed knowledge of the actual load, rather than just knowledge of the best and worst case loads. Second, even if one could control nonextremal measures of packet delay for the aggregate traffic in the network, this does not control the value of such measures for individual flows; e.g., the average delay observed by a particular flow need not be the same as, or even bounded by, the average of the aggregate (see [27] for a discussion of related issues). Thus, controlling nonextremal measures of delay for the aggregate is not sufficient, and we judge it impractical to control nonextremal measures of delay for each individual flow. Third, as will be argued in the next section, applications that require quantitative delay bounds are more sensitive to the extremes of delay than the averages or other statistical measures, so even if other delay related qualities of service were practical they would not be particularly useful. We discuss this in the section below when we discuss real-time applications.

Why have we not included bandwidth as a quality of service about which the network makes commitments? This is primarily because, for applications which care about the time-of-delivery of each packet, the description of per-packet delay is sufficient. The application determines its bandwidth needs, and these needs are part of the traffic characterization passed to the network's admission control algorithm; it is the application which then has to make a commitment about the bandwidth of its traffic (when requesting a quantitative service commitment from the network), and the network in turn makes a commitment about delay. However, there are some applications which are essentially indifferent to the time-of-delivery of individual packets; for example, when transferring a very long file the only relevant measure of performance is the finish time of the transfer, which is almost exclusively a function of the bandwidth. We discuss such applications at the end of Section 3.3.

3.2 Application Delay Requirements

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

[2]For those links where this is not a good approximation, such as some wireless links, we expect there to be hop-by-hop error recovery so that at the network level there is a low error rate.

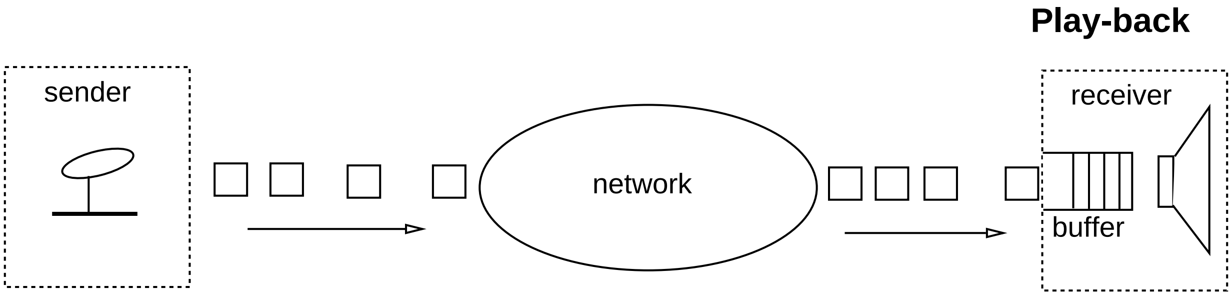

Figure 1: A schematic diagram of a playback application. The signal is generated and packetized at the sender and then transmitted over the network. The receiver, in order to remove the effects of network-induced delay jitter, buffers the packets until their playback points.

consider the delay requirements of these two classes separately. For the purposes of the discussion that follows, we assume that all applications involve point-to-point communication, with all packets requiring the same service. At the end of Section 3.3 we discuss the case of multipoint-to-multipoint communication. In Section 5 we address the case where some packets in a flow are more important than others.

3.2.1 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; Figure 1 illustrates such an application. 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. This variation in delay has traditionally been called "jitter". The receiver depacketizes the data and then attempts to faithfully play back the signal. This is done by buffering the incoming data to remove the network induced jitter 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[3]. For the purposes of discussion, let us temporarily assume that such playback applications have some intrinsic data generation process that is unalterable; later in this section we will return to this point.

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. In general, latency is the delay between the two (or more) ends of a distributed application; for

[3]It is an oversimplification to say that the data is useless; we discuss below that a receiving application could adjust the playback point as an alternative to discarding late packets.

playback applications, latency is the delay between the time the signal is generated at the source and the time the signal is played back at the receiver, which is exactly the offset delay. Applications vary greatly in their sensitivity to latency. 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 value of the offset delay; other playback applications, such as transmitting a movie or lecture, are not.

Fidelity is the measure of how faithful the playback signal is to the original signal. The playback signal is incomplete when packets arrive after their playback point and thus are dropped rather than played back. The playback signal becomes distorted when the offset delay is varied. Therefore, fidelity is decreased whenever the offset delay is varied and whenever packets miss their playback point. 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 [4]. Intolerance to loss of fidelity might arise because of user requirements (e.g., distributed symphony rehearsal), or because the application hardware or software is unable to cope with missing pieces of data. On the other hand, users of tolerant applications, as well as the application hardware and software, are prepared to accept occasional distortions in the signal. We expect that the vast bulk of audio and video applications will be tolerant.

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. In contrast, tolerant applications need not set their offset delay greater than the absolute maximum delay, since they can tolerate some late packets. Moreover, tolerant applications can vary the offset delay to some extent, as long as it doesn't create too much distortion.

Thus, tolerant applications have a much greater degree of flexibility in how they set and adjust their offset delay. In particular, 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 (a more precise term is *delay-adaptive* playback applications, to distinguish it from the *rate-adaptive* playback applications we discuss below). This adaptation amounts to gambling that the past packet delays are good predictors of future packet delays; when the application loses the gamble there is a momentary loss of data as packets miss their playback

[4] Obviously, applications lie on a continuum in their sensitivity to fidelity. Here we are merely considering two cases as a pedagogical device to motivate our service model, which indeed applies to the full spectrum of applications.

points, but since the application is tolerant of such losses the decreased offset delay may be worth it. Besides the issue of inducing late packets, there is a complicated tradeoff between the advantage of decreased offset delay and the disadvantage of reduced fidelity due to variations in the offset. Thus, how aggressively an application adapts, or even if it adapts at all, depends on the relative ergonomic impact of fidelity and latency. Our main observation here, though, is that by adapting to the delays of incoming packets, tolerant playback applications can often profit by reducing their offset delay when the typical delays are well below the absolute maximum; this advantage, of course, is accompanied by the risk of occasional late packets.

So far we have assumed that an application's data generation process is unalterable. 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. 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.

We now state several of our assumptions about the nature of future real-time applications. First, we believe that most audio and video applications will be playback applications, and we therefore think that playback applications will be the dominant category of real-time traffic. By designing a service model that is appropriate for these playback applications, we think we will have satisfactorily (but perhaps not optimally) met the needs of all real-time applications. Second, we believe that the vast majority of playback applications will be tolerant and that many, if not most, of these tolerant playback applications will be adaptive. The idea of adaptive applications is not relevant to circuit switched networks, which do not have jitter due to queueing. Thus, most real-time devices today, like voice and video codecs, are not adaptive. Lack of widespread experience may raise the concern that adaptive applications will be difficult to build. However, early experiments suggest that it is actually rather easy. Video can be made to adapt by dropping or replaying a frame as necessary, and voice can adapt imperceptibly by adjusting silent periods. In fact, such adaptive approaches have been employed in packetized voice applications since the early 70's (see [8, 36]); the VT [1], NEVOT [31], and VAT [24] packet voice protocols, which are currently used to transmit voice on the Internet, are living examples of such adaptive applications.

Third, we believe that most playback applications will have sufficient buffering to store packets until their playback point. We base our belief on the fact that the storage needed is a function of the queueing delays, not the total end-to-end delay. There is no reason to expect that queueing delays for playback applications will increase as networks get faster (in fact, for an M/M/1 queueing system with a fixed utilization, queueing delays are inversely proportional to the speed), and it is certainly true that memory is getting cheaper, so providing sufficient buffering will become increasingly practical. Fourth, and last, we assume that applications have sufficient knowledge about time to set the playback point. The notion of a playback application implies that such applications have some knowledge about the original generation time of the data. This knowledge could either be explicitly contained in timestamps, or an approximation could be implicitly obtained by knowing the inter-packet generation intervals of the source.

3.2.2 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 tend to depend more on the average delay than on the tail of the 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.

Some elastic applications, like Telnet, have an intrinsic data generation process which is largely independent of network conditions. However, there are many elastic applications, in particular those involving bulk transfer, which can alter their packet transmission process.

3.3 Delay Service Models

We now turn to describing service models that are appropriate for the various classes of applications that were discussed in the previous paragraphs. Since we are assuming that playback applications comprise the bulk of the real-time traffic, we must design service models for intolerant playback applications, tolerant playback applications (which can be either adaptive or non-adaptive), rate-adaptive playback applications (which can be either tolerant or intolerant), and elastic applications.

The offset delay of intolerant playback applications must be no smaller than the maximum packet delay to achieve the desired faithful playback. Furthermore, this offset delay must be set before any packet delays can be observed. Such an application can only set its offset delay appropriately if it is given a perfectly reliable[5] 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. Note that the delay bound not only allows the application to set its offset delay appropriately, but it also provides the information necessary to predict the resulting latency of the application.

Since such an intolerant playback application will queue all packets until their respective playback points, application performance is completely independent of when the packets arrive, as long as they arrive within the delay bound. The fact that we assume that there is sufficient buffering means that we need not provide a nontrivial lower bound to delay; only the trivial *no-queueing* minimum

[5] By perfectly reliable, we mean that the bound is based on worst case assumptions about the behavior of all other flows. The validity of the bound is predicated on the proper functioning of all network hardware and software along the path of the flow.

delay will be given as part of the service specification.

A tolerant playback application which is not adaptive will also need some form of a delay bound so that it can set its offset delay appropriately. Since the application is tolerant of occasional late packets, this bound need not be perfectly reliable. For this class of applications we propose a service model called *predictive service* which supplies a fairly reliable, but not perfectly reliable, delay bound. For this service, the network advertises a bound which it has reason to believe with great confidence will be valid, but cannot formally "prove" its validity[6]. 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 and lower realized delays[7].

It is important to emphasize that this is *not* a statistical bound, in that no statistical failure rate is provided to the application in the service description. We do not think it feasible to provide a statistical characterization of the delay distribution because that would require a detailed statistical characterization of the load. We do envision the network ensuring the reliability of these predictive bounds, but only over very long time scales; for instance, the network could promise that no more than a certain fraction of packets would violate the predictive bounds over the course of a month[8]. Such a statement is not a prediction of performance but rather a commitment to adjust its bound-setting algorithm to be sufficiently conservative.

All nonadaptive applications, whether tolerant or not, need an *a priori* delay bound in order to set their offset delay; the degree of tolerance only determines how reliable this bound must be. In addition to being necessary to set the offset delay, these delay bounds provide useful estimates of the resulting latency. Nonadaptive tolerant applications, like the intolerant applications considered above, are indifferent to when their packets arrive, as long as they arrive before the delay bound.

Recall, however, that we are assuming that many, if not most, tolerant playback applications are adaptive. Thus, we must design the service model with such adaptation in mind. Since these applications will be adapting to the actual packet delays, a delay bound is not needed to set the offset delay. However, in order to choose the appropriate level of service, applications need some way of estimating their performance with a given level of service. Ideally, such an estimate would depend on the detailed packet delay distribution. We consider it impractical to provide predictions or bounds on anything other than the extremal delay values. Thus, we propose offering the same predictive service to tolerant adaptive applications, except that here the delay bound is

[6]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.

[7]For nonadaptive applications, the realized latency is lower with predictive service since the fairly reliable bounds will be less conservative than the perfectly reliable bounds of guaranteed service. For adaptive applications, as we discuss below, the minimax component of predictive service can, and we expect usually will, reduce the average latency, i.e. the average value of the offset delay, to be well below the advertised bound.

[8]Such an assurance is not meaningful to an individual flow, whose service over a short time interval might be significantly worse than the nominal failure rate. We envision that such assurances would be directed at the regulatory bodies which will supervise the administration of such networks. However, we should note that there may very well be pricing schemes which refund money if the service delivered to an individual application doesn't meet some standard (such as a given fraction of packets obey the delay bound); this is not a service commitment but rather a monetary one.

not primarily used to set the offset delay (although it may be used as a hint) but rather is used to predict the likely latency of the application.

The actual performance of adaptive applications will depend on the tail of the delay distribution. We can augment the predictive service model 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 distribution. Here the fairly reliable predictive delay bound is the quantitative part of the service commitment, while the minimax part of the service commitment is a relative service commitment. We could offer separate service models for adaptive and nonadaptive tolerant playback applications, with both receiving the predictive service as a quantitative service commitment and with only adaptive applications receiving the minimax relative commitment. However, since the difference in the service models is rather minor, we choose to only offer the combination of predictive and minimax service.

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[9]; 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.

As we discussed above, both of these service models have a quantitative component. In order to offer this service, 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. This characterization will not be a detailed statistical characterization (we do not believe applications will be able to provide those) but instead will be a worst-case characterization; the flow will commit to not exceed some usage envelope or filter (e.g., a token bucket or leaky bucket). A fundamental point of our overall architecture is that traffic characterization and admission control are necessary for these real-time delay bound services. For rate-adaptive applications, these traffic characterizations are not immutable. We can thus augment the service model by allowing the network to notify (either implicitly through packet drops or explicity through control packets) rate-adaptive applications to change their traffic characterization. We will discuss this more thoroughly in Section 5.

The fourth category for which we must develop a service model is elastic applications. Elastic applications are rather different than playback applications; while playback applications hold packets until their playback time, elastic applications use the packet whenever it arrives. Thus, reducing the delays of any packet tends to improve performance. Furthermore, since there is no offset delay, there is no need for an *a priori* characterization of the delays. An appropriate service model is to provide *as-soon-as-possible*, or ASAP service, which is a relative, not quantitative, commitment[10]. Elastic applications vary greatly in their sensitivity to delay (which, as we mentioned earlier, is

[9]Efficiency can be thought of as the number of applications that can be simultaneously serviced with a given amount of bandwidth; for a fuller definition, see [6, 33].

[10]We choose not to use the term "best-effort" for the ASAP service since that connotes the FIFO service discipline.

probably more a function of the average delay than of the maximum delay), and so the service model for elastic traffic should distinguish between the various levels of delay sensitivity. We therefore propose a multiclass ASAP service model 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, this service model does not provide any quantitative service commitment, and thus applications cannot predict their likely performance and are also not subject to admission control. However, we think that rough predictions about performance, which are needed to select a service class, could be based on the ambient network conditions and historical experience. If the network load is unusually high, the delays will degrade and the users must be prepared to tolerate this, since there was no admission control to limit the total usage.

However, there may be some cases where an application (or the user of the application) might want to know more precisely the performance of the application in advance. For instance, a Telnet user might want to ensure that the delays won't interfere with her typing. For these cases, the application can request predictive service (since the firmness of the guaranteed bound is probably not required) provided it is willing to specify the maximum transmission rate desired. Note that since the network will then require compliance with the advertised transmission rate, the application cannot get a higher throughput rate than what it requested.

There are two issues regarding the elastic service model[11] that we do not address in this memo, and propose that these issues be revisited once the rest of the core service model is defined. First, there is the issue of relative treatment of flows. One could treat each elastic packet independently, and allocate service based merely on time-of-arrival and the level of ASAP service requested. Alternatively, one could also attempt to control the aggregate resources used by each individual flow, such as is done in the Fair Queueing service model as described in [7]. We do not address the relative treatment of various flows at this time, since it will not affect the basic service interface. Second, there is the issue of feedback. As we noted before, some elastic applications can adjust their transmission pattern. This adjustment can be in response to implicit signals, such a packet drops or delay, or explicit signals such as congestion bits in the packet header or separate control packets. Again, we do not address at this time the form or content of such feedback signals since they do not affect the basic service interface.

At the beginning of this section, we made the initial assumption that delay was the only quality of service about which the network needed to make commitments. We now revisit this issue and ask if that is indeed the case. For the typical real-time or elastic application which cares about the delays of individual packets, there seems to be no need to include any other quality of service. However, we observed earlier that there are some applications, such as transfers of very long files, which are essentially indifferent to the delays of individual packets and are only concerned with overall delay of the transfer. For these *indifferent* applications, bandwidth rather than delay is a more natural characterization of the desired service, since bandwidth dictates the application performance. If such an application has no intrinsic overall delay requirement, then the desired service is to finish the transfer as quickly as possible. The desired service is *as-much-bandwidth-*

[11] We have used the convenient, but perhaps confusing convention, of referring to elastic service and real-time service when in fact the terms real-time and elastic refer to a class of applications.

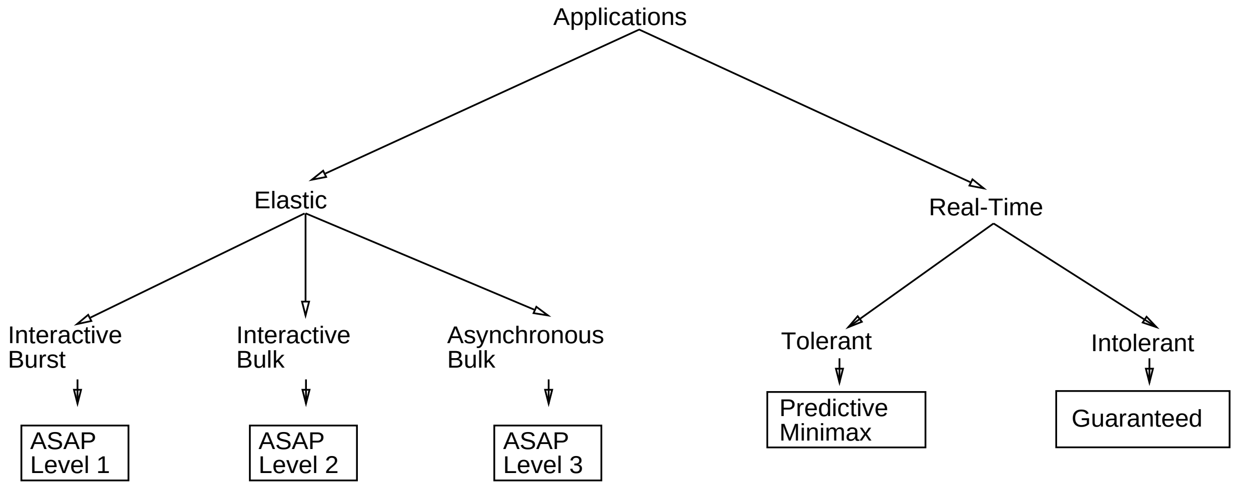

Figure 2: Our rough taxonomy of applications and their associated service models. We have arbitrarily depicted three levels of ASAP service.

as-possible. By servicing packets as soon as possible, the ASAP service described above delivers exactly this *as-much-bandwidth-as-possible* service. Thus, while we did not explicitly consider bulk transfer applications, our proposed service model already provides the desired service for bulk transfer applications with no intrinsic overall delay requirements.

However, if this bulk transfer application had some intrinsic overall delay requirement, i.e. it required the transfer to be completed within a certain time, then the ASAP service is no longer sufficient. Now, the appropriate service is to allow the application to request a specified amount of bandwidth; the application chooses this bandwidth amount so that the transfer will be completed in time. An application can secure a given amount of bandwidth through either of the real-time services. The per-packet delay bounds provided by these real-time services are superfluous to bulk transfer applications with overall delay requirements. While one could imagine a different service which provided a commitment on bandwidth but not per-packet delay, the difference between requesting a large delay bound and no delay bound is rather insignificant, and thus we expect that such indifferent applications with delay requirements will be adequately served by predictive service with very large delay bounds. This has the disadvantage that indifferent applications with delay requirements do not get as-much-bandwidth-as-possible, but are constrained to their reserved amount.

Figure 2 depicts our taxonomy of applications and the associated service models. This taxonomy 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 cost, fidelity, and latency. 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.

We have defined the core service model in terms of point-to-point flows. We can easily generalize this service model to incorporate multipoint-to-multipoint flows. Clearly for elastic traffic there is no change to the service model. For the real-time service models, the delay bounds are specified for each point-to-point pair, and the traffic characterizations apply to the sum of the flow's traffic at each hop along the flow's path.

4 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. Our reasoning for this is as follows. Meeting individual application service needs is the task of quality of service commitments; however, both the number of quantitative service commitments that can be simultaneously made, and the quantitative performance delivered by the relative service commitments, depend on the aggregate bandwidth. Thus, when considering collective entities we claim that we need only control the aggregate bandwidth available to the constituent applications; we can deal with all other performance issues through quality of service commitments to individual flows. Embedded within this reasoning is the assumption that bandwidth is the only scarce commodity; if buffering in the switches is scarce then we must deal with buffer-sharing explicitly, but we contend that switches should be built with enough buffering so that buffer contention is not the primary bottleneck.

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; however, one must be careful to state exactly what this means. The following example will highlight some of the policy issues implicit in link-sharing. Consider three firms, 1, 2, and 3, who respectively have shares 1/4, 1/4, and 1/2 of some link. Assume that for a certain hour, firm 1 sends no traffic to the link while firms 2 and 3 each send enough to use the entire capacity of the link. Are firms 2 and 3 restricted to only using their original shares of the link, or can they use firm 1's unused bandwidth? Assume for now that they are allowed to use firm 1's unused bandwidth. Then, how is firm 1's share of the link split between firms 2 and 3? If, in the next twenty minutes, all three firms each send enough traffic to consume the entire link, is the link allocated solely to firm 1 in order to make up for the imbalance in aggregate bandwidth incurred during the first hour, or is the link shared according to the original shares? Thus, there are three policy questions to be resolved: can firms use each other's unused bandwidth, how is this unused bandwidth allocated to the remaining firms, and over what time scale is the sharing of bandwidth measured? Clearly the answer to the first question must be affirmative, since much of the original motivation for link-sharing is to take advantage of the economies of statistical aggregation. As for the second question, one can imagine many rules for splitting up the excess bandwidth but here we propose that the excess is assigned in proportion to the original shares so that in the above example during the first hour the link would be split 1/3, 2/3 for firms 2 and 3 respectively. The answer to the third question is less clear. The preceding example indicates that if sharing is measured over some time scale T then a firm's traffic can be halted for a time on the order of T under certain conditions; since such cessation should be avoided, we propose doing the sharing on an instantaneous basis (i.e., the limit of T going to zero). This would dictate that during this next twenty minutes the bandwidth is split exactly according to the original shares: 1/4, 1/4, and 1/2. This policy embodies a "use-it-or-lose-it" philosophy in that the firms are not given credit at a later date for currently unused bandwidth.

An idealized fluid model of instantaneous link-sharing with proportional sharing of excess is the fluid processor sharing model (introduced in [7] and further explored in [28, 29]) 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. More specifically, we let μ be the speed of the link and we give each entity i its own virtual queue which stores its packets as they await service. For each entity i we define the following quantities: s_i, the share of the link; $c_i(t)$, the cumulative number of bits in the traffic stream that have arrived by time t; and the backlog $b_i(t)$, the number of bits remaining in the virtual queue at time t. Whenever a real packet arrives at the switch belonging to entity i, we place a corresponding idealized packet at the tail of that entity's virtual queue. The service within each such virtual queue is FIFO. We now describe how service is allocated among the different virtual queues. The idealized service model is defined by the equations:

$$b_i'(t) = c_i' - \min[s_i\lambda, c_i'] \qquad if\ b_i(t) = 0 \tag{1}$$

and

$$b_i'(t) = c_i'(t) - s_i\lambda \qquad if\ b_i(t) > 0 \tag{2}$$

where $b_i'(t)$ and $c_i'(t)$ denote the time derivatives of $b_i(t)$ and $c_i(t)$, and where λ is the unique constant that makes $\sum_i b_i' = \mu - \sum_i c_i'$ (when no such value exists, we set $\lambda = \infty$).

At every instant the excess bandwidth, that is the bandwidth left over from flows not using their entire share of bandwidth, is split among the active entities (i.e., those with $b_i > 0$) in proportion to their shares; each active[12] entity receives an instantaneous bandwidth that is greater than or equal to their share of the full transmission rate.

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. The link-sharing service model provides quantitative service commitments on bandwidth shares that the various entities receive.

Heretofore we have considered link-sharing across a set of entities with no internal structure to the entities themselves. However, the various sorts of link-sharing requirements presented above could conceivably be nested into a hierarchy of link-sharing requirements, an idea first proposed by Jacobson and Floyd [23]. 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. We propose extending the idealized link-sharing service model presented above to the hierarchical case. The policy desires will be represented by a tree with shares assigned to each node; the shares belonging to the children of each node must sum to the share of the node, and the top node represents the full link and has a unit share. Furthermore, each node has an arrival stream described by $c_i(t)$ and a backlog $b_i(t)$ with the quantities of the children of each node summing to the quantity of the node. Then, at each node we invoke the fluid processor sharing model among the children, with the instantaneous link speed at the i'th node, $\mu_i(t)$, set equal to the rate $b_i'(t)$ at which bits are draining out of that node's virtual queue. We can start this model at the top node; when propagated down to the leaf nodes, or bottom-level entities, this determines the idealized service model.

The introduction of a hierarchy raises further policy questions which are illustrated by the following example depicted in Figure 3. Consider two firms, 1 and 2, each with two protocols, 'a' and 'b'. Let us assume that each of the bottom-level entities, 1a, 1b, 2a and 2b, has a 1/4 share of the link. When all of the bottom-level entities are sending enough to consume their share, the bandwidth is split exactly according to these shares. Now assume that at some instant there is no offered 2b traffic. Should each of 1a,1b and 2a get 1/3 of the link, or should 1a and 1b continue to get 1/4,

[12] There are three states a flow can be in: active ($b_i > 0$), inactive ($b_i = 0$ and $c_i' = 0$), and in-limbo ($b_i = 0$ but $c_i' > 0$).

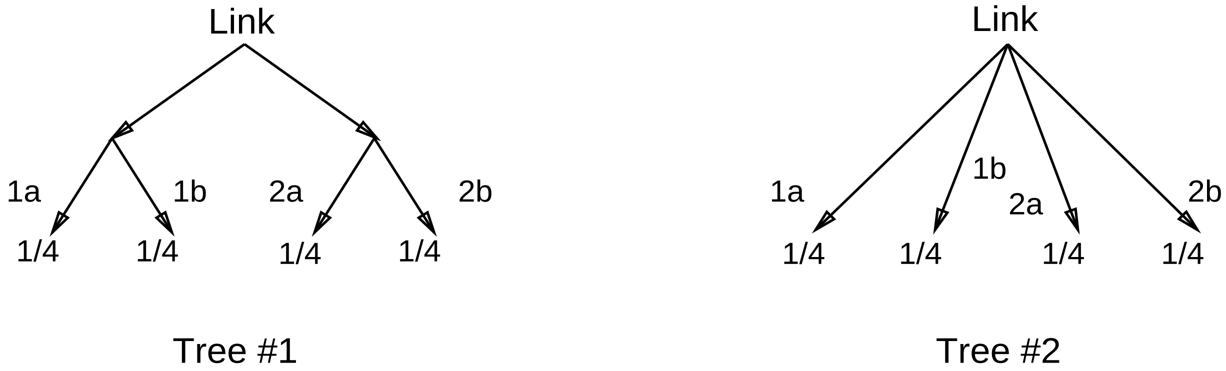

Figure 3: Two possible sharing trees with equal shares at all leaf nodes. When one of the leaf nodes is not active, the trees produce different bandwidth shares for the remaining active nodes.

with 2a getting the remaining 1/2 share of the link which is the total of the shares belonging to firm 2? This is a policy question to be determined by the firms, so the service model should allow either. Figure 3 depicts two possible sharing trees. Tree #1 in the figure produces the 1/4, 1/4, 1/2 sharing whereas tree #2 produces the 1/3, 1/3, 1/3 sharing. When the link-sharing service commitment is negotiated, it will be specified by a tree and an assignment of shares for the nodes.

In the hierarchical model, the bandwidth sharing between the children of a given node was independent of the structure of the grandchildren. One can think of far more general link-sharing service models. Assume that in the example above that protocol 'a' carries traffic from applications with tight delay requirements and protocol 'b' carries traffic from applications with loose delay requirements. The two firms might then want to implement a sharing policy that when 1a is not fully using its share of the link, the excess is shared equally among 1b and 2a, but when 1b is not fully using its share of the link we will give the excess exclusively to 1a. To implement this more complicated policy, it is necessary to take the grandchildren structure into account. We think that this sort of flexibility is probably not needed, for the same reason that we restricted ourselves to bandwidth as the only collective concern; quality of service issues should be addressed via quality of service commitments and not through the link-sharing service model. For this same reason, we do not make priority distinctions between the various nodes, but merely allocate shares of bandwidth. Therefore, for our resource-sharing service model we restrict ourselves to the hierarchical service model presented above.

In Section 3 we 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 bottom-level entity, admission control must keep the cumulative guaranteed and predictive traffic from exceeding the assigned link-share.

5 Denial of Service

To meet its quantitative service commitments, the network must employ some form of admission control. Without the ability to deny flows admission to the network, one could not reliably provide the various delay bound services offered by our service model. In fact, admission control is just one aspect of denial of service; there are several other ways in which service can be denied. Denial of service, in all of its incarnations, plays a fundamental role in meeting quantitative service commitments. In particular, denial of service can be used to augment the resource sharing portion of the core service model by supporting utilization targets. Moreover, denial of service, through the use of the preemptable and expendable service options discussed below, can enable the network to meet its service commitments while still maintaining reasonably high levels of network utilization.

Denial of service, like service commitments, can occur at various levels of granularity. Specifically, denial of service can apply to whole flows, or to individual packets within a flow. We discuss these two cases separately.

5.1 Denial to Flows

Denial of service to a flow can occur either before or during the lifetime of that flow. Denying service to a flow before it enters the network is typically referred to as admission control. As we envision it, in order to receive either of the two real-time bounded delay services (guaranteed and predictive), a flow will have to explicitly request that service from the network, and this request must be accompanied by a worst-case characterization of the flow's traffic stream. This characterization gives the network the information necessary to determine if it can indeed commit to providing the requested delay bounds. The request is denied if the network determines that it cannot reliably provide the requested service. References [12, 17, 21, 25] discuss various approaches to admission control.

In addition, a service model could offer a *preemptable* flow service, presumably for a lower cost than non-preemptable service. When the network was in danger of not meeting some of its quantitative service commitments, or even if the network was merely having to deny admission to other flows, then it could exercise the "preemptability option" on certain flows and immediately discontinue service to those flows by discarding their packets (and, presumably, sending a control message informing those flows of their termination). By terminating service to these preemptable flows, the service to the flows that are continuing to receive service will improve, and other non-preemptable flows can be admitted.

Recall that rate-adaptive flows are able to adjust their transmission rate. For these flows we can offer an *adjustable* flow service, again presumably for a lower cost than the regular non-preemptable, non-adjustable service. When the network was in danger of not meeting some of its quantitative service commitments, or even if the network was merely having to deny admission to other flows, then it could exercise the "adjustability option" of these flows and request that they reduce their transmission rate. Similarly, when the network had spare capacity, it could inform these flows that they could increase their transmission rate.

Admission control can be used to augment the link-sharing service model described in the previous section. Link-sharing uses packet scheduling to provide quantitative service commitments about bandwidth shares. This service is designed to provide sharing between various entities which have explicitly contracted with the network to manage that sharing. However, there are other collective policy issues that do not involve institutional entities, but rather concern overall utilization levels of the various service classes (guaranteed, predictive, ASAP). Because they are not explicitly negotiated, and so no service commitments are at stake, these utilization levels are not controlled by packet scheduling but instead are controlled by the admission control algorithm. All real-time flows are subject to scrutiny by the admission control process; only those flows that are accepted can use the network. If the admission control algorithm used the criteria that a flow was accepted if and only if it could be accepted without violating other quality of service commitments, then the utilization levels of the various classes will depend crucially on the order in which the service requests arrived to the network. One might desire, instead, to make explicit policy choices about these various level of utilization. For instance, it is probably advisable to prevent starvation of any particular class of traffic; an explicit control would be needed to prevent starvation of elastic traffic since the ASAP service does not involve resource reservation. In addition, one might want the admissions process to ensure that requests for large amounts of bandwidth were not always squeezed out by numerous smaller requests.

To prevent such problems, we must introduce some guidelines, called *utilization targets*, into the admission control algorithm so that the utilization levels are not just dependent on the details of the load pattern but instead are guided towards some preferred usage pattern. This utilization target service model involves only admission control; thus, it is not properly part of the core service model. We mention utilization targets here because other aspects of the core service model rely on these utilization targets, and also because it is so similar to the link-sharing model, in that it represents policy objectives for aggregated classes of traffic.

5.2 Denial To Packets

While denial of service is usually associated with admission control, it also can be performed on a packet-by-packet granularity. Denial of service to individual packets could occur by means of a *preemptable* packet service, whereby flows would have the option of marking some of their packets 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, the delays of the not-preempted packets will be reduced.

The basic idea of allowing applications to mark certain packets to express their "drop preference" and then having the network discard these packets if the network is congested has been circulating in the Internet community for years, and has been simulated in Reference [32]. The usual problem in such a scheme is defining what congestion means. In the Internet, with its simple service model, one usually equates congestion with the presence of a sizable queue. However, this is a network-centric definition that is not directly related to the quality of service desired by the various applications. In contrast, in our setting, we can make a very precise definition of congestion that is directly tied to the applications' service requirements: congestion is when some of the quantitative service commitments are in danger of being violated. The goal of admission control is to ensure that this situation arises extremely infrequently.

The basic idea of preemptability can usefully be extended in two directions. First, for the purposes of invoking the preemptability options, one can stretch the definition of a quantitative service commitment to include implicit commitments such as compliance with the historical record of performance. That is, one could choose to drop packets to make sure that the network continued to provide service that was consistent with its past history, even if that past history was never explicitly committed to. Furthermore, one could also extend the definition of a quantitative service commitment to the utilization targets discussed above.

Second, one can define a class of packets which are 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 will 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. Such expendable packets could be dropped not only when quantitative service commitments are in danger of being violated, but also when implicit commitments and utilization targets, as described above, are in danger of being violated.

The goal of these preemptable and expendable denial of service options (both at the packet and flow level of granularity) is to identify and take advantage of those flows that are willing to suffer some interruption of service (either through the loss of packets or the termination of the flow) in exchange for a lower cost. The preemptable flows and packets provide the network with a margin of error, or a *cushion*, for absorbing rare statistical fluctuations in the load. This will allow the network to operate at a higher level of utilization without endangering the service commitments made to those flows who do not choose preemptable service. Similarly, expendable packets can be seen as *filler* for the network; they will be serviced only if they do not interfere with any service commitment but there is no expectation that their being dropped is a rare event. This will increase the level of utilization even further. We will not specify further how these denial of service, or preemptability, options are defined, but clearly there can be several levels of preemptability, so that an application's willingness to be disrupted can be measured on more than a binary scale.

6 Alternative Viewpoints

In this section, we discuss several other viewpoints on the problem of providing integrated services.

6.1 Scheduling Algorithms vs. Service Models

The motivating principle of this memo is that the service model is primary. However, one could contend that because we do not yet know the service needs of future applications, the most important goal is to design flexible and efficient packet scheduling implementations. Obviously both packet scheduling implementations and service models are tremendously important, but the debate here is over which one should guide the design of the Internet. There are three points to be made.

First, the service model must be made explicit to application designers. Currently, there are a rather limited number of network-intensive applications; the network can, to a large extent, determine the service requirements of a packet by inspecting the port number. However, as the variety of network-intensive applications increases, and as the service requirements of these applications begin to depend on the user's personal demands (e.g., high and low priority mail, high and low quality video from the same codec, etc.), port numbers will no longer be sufficient to identify service requirements. Rather than having the network implicitly deliver the appropriate service, the applications will have to explicitly request the desired service. For this to happen, the service model must be made explicit (so that application designers know about it), and it obviously must remain relatively stable; the service model should not just be implicitly defined by the packet scheduling implementation. Thus, regardless of whether the packet scheduling algorithm is flexible or not, the service model must be made explicit and remain relatively stable.

Second, there is a fundamental difference in the time-scale over which packet scheduling implementations and service models have impact. Once a router vendor with a substantial market presence adopts a new packet scheduling implementation, it will likely remain fixed for several years. So, in the short term, we need to ensure that such packet scheduling implementations embody enough flexibility to adapt if a new service model is adopted, or the current service model is extended, during the product's lifetime. However, router technology, and the embedded packet scheduling implementations, do evolve as new products are introduced, and so one cannot expect that packet scheduling implementations will remain fixed for many years. On the other hand, the time scale of service models is rather different. It typically takes much longer for a new service model to become adopted and utilized, because it must be embedded in user applications. However, once a service model does become adopted it is much harder to change, for precisely the same reason. Thus, we can say that while the set of packet scheduling implementations will likely freeze first, the service model freezes harder. For this reason we choose to focus on the service model.

Third, the role of flexibility must be clarified. The services offered to individual flows by a packet scheduling algorithm must be part of a service model and, as we argued above, the service model does not change rapidly (except in experimental networks, where perhaps using flexible and efficient packet scheduling implementations is important); in particular, we expect service models to change much less rapidly than packet scheduling algorithms. Thus, for quality of service commitments to individual flows, flexibility is not of great importance. However, the link-sharing portion of the service model is not exercised by individual applications but rather by network managers through some network management interface. This portion of the service model can change much more rapidly, so flexibility is indeed important for link-sharing and other forms of resource sharing. The debate over the relative importance of service models and packet scheduling implementations reflects, at least in part, a deeper disagreement over the extent to which quality of service needs are met indirectly by link-sharing, which controls the aggregate bandwidth allocated to various collective entities, as opposed to being met directly by quality of service commitments to individual flows. Actually, the important distinction here is not between link-sharing and delay related services, but rather between those services which require explicit use of the service interface, and those that are delivered implicitly (i.e., based on information automatically included in the packet header such as port numbers). Network architectures designed around such implicit quality of service mechanisms do not require a well-defined service model; the network architecture we have advocated involves explicit quality of service mechanisms and therefore requires a stable service model.

6.2 Why Use Admission Control?

Real-time service plays a central role in the proposed service model. We should note that there is another viewpoint on this issue, which has not yet been adequately articulated in the literature. It is conceivable that the combination of adaptive applications and sufficient overprovisioning of the network could render such delay bounds, with the associated need for admission control, unnecessary; applications could adapt to current network conditions, and the overprovisioning would ensure that the network was very rarely overloaded[13]. In this view, it would be sufficient to provide only the several classes of ASAP service without *any* real-time services. The first question is, can one indeed overprovision the network so that it is extremely rarely overloaded? It is true that the statistical demand in the phone network is well characterized, and overprovisioning has become a finely honed art. However, there are three crucial differences between the phone network and the Internet which lead us to the conclusion that the extreme variability of the offered load will require too great a degree of overprovisioning to make this approach practical. First, we do not expect the usage patterns on the Internet to be nearly so well characterized. In the phone network the usage patterns tend to revolve around human behavior, which changes rather slowly. However, in the Internet, the transfer of a few file repositories can create a dramatic and immediate shift in traffic patterns. Second, the variability in usage of an individual phone user is quite limited. In contrast, computer network usage can easily vary by three orders of magnitude, from 64kbps voice to 100mbps HDTV. Even if the law of large numbers does apply, the intrinsic variance of the individual distributions means that the resulting variance of the aggregate usage will be several orders of magnitude bigger than in the phone network. Third, regardless of price, there are natural intrinsic limits to the maximum bandwidth demand on the phone network: every person and FAX machine placing a call simultaneously. In contrast, there is no reason to expect that if bandwidth were sufficiently cheap there would be limits to the bandwidth consumption on the Internet (think of having video-on-demand everywhere). Thus, unless we use excessively high prices to artificially lower demand, we doubt we can overprovision the network so that it is extremely rarely overloaded [14]. This issue then reduces to choosing either to lower demand through high prices or to occasionally turn demand away when in an overload; we think it far preferable to encourage network use by keeping prices low and then use admission control to ration when demand exceeds supply.

Given that overloads will occur if no admission control is used, the second question is: can applications adequately adapt to these overloaded conditions, or should we use admission control to prevent these overloads from occurring? Even if one assumes that adaptation is done instantaneously (so that there are no transient periods where the offset delays are incorrectly set), there is the basic question of whether the user population would be happier all sharing an overloaded network, or would they prefer having some users turned away. For typical elastic applications such as Telnet, it is most likely preferable to share the overloaded network. For typical real-time applications such as remote interactive video, we conjecture that it is preferable to turn some users away because the rapid increase in delays and packet drops as a function of load causes severe degradation of application performance even for adaptive applications. In short, the ability to adapt to worse conditions does not mean that applications are unaffected by these conditions. For this reason we conclude that admission control is necessary.

---

[13] Of course, this viewpoint is predicated on the nonexistence of applications which have hard real-time requirements.

[14] Using prices in this way will likely cause the excess demand to bypass the Internet and turn to alternate providers.

A common counterargument to our line of reasoning is that users will be unhappy with any network that denies service with any significant frequency, and so we are merely trading off the unhappiness with overloading for the unhappiness caused by denial of service. While users may expect very low rates of denial for low-bandwidth applications like voice, there will not likely be the same expectation for extremely bandwidth intensive applications like HDTV. We expect that it will be rather easy, and fairly efficient (i.e., result in a reasonably high utilization level), to provision the network so that it can easily accept almost all phone calls, but will occasionally turn away much larger bandwidth requests.

6.3 Variations on the Service Model

There are other approaches to defining real-time service. The real-time service advocated here provides a bound on the maximum delay of packets, provided that the application's traffic load conforms to some prearranged filter. One could provide not only a bound on the maximum delay but also a nontrivial bound (i.e., a bound other than the no-queueing bound) on the minimum delay. We did not include such nontrivial lower bounds on delay in our present service model because they serve only to reduce buffering at the receiver and we do not expect buffers to be a bottleneck; furthermore, if some applications do need additional buffering, this can easily be supplied at the edge of the network and need not be built into the basic core service model.

A rather different form of service model is to offer statistical characterizations of performance. We explicitly reject such statistically characterized service offerings because they inherently require a statistical characterization of individual flows (or at least of the aggregate traffic), and we doubt that such characterizations will be available. Instead, we rely only on worst-case characterizations of the flows.

Finally, one can define different link-sharing service models; in particular, as discussed in [13], one can incorporate priorities between entities into the link-sharing service model (the model presented here does include priorities in a single entity's traffic, but not between entities). We do not include this feature for two reasons. First, a basic principle of this service model is that the quality of service requirements of individual applications should be addressed primarily through explicit service requests. Second, and much more importantly, the priority features will not produce dramatically different delay behaviors unless the traffic is very close to the bandwidth limits imposed by link-sharing.

7 Acknowledgments

We would like to acknowledge that the thoughts discussed in this memo reflect the contributions of many others. In particular, the works of Parekh and Gallager [28, 29], Ferrari et al. [10, 11, 12, 35], Jacobson and Floyd [2, 23, 13], Golestani [14, 15], Guerin et al. [16, 17], Kurose et al. [4, 18, 27, 32, 37], Lazar et al. [19, 20, 21, 22], and Kalmanek et al. [26] have been critical in shaping our thinking on this matter. Discussions with our ISIP collaborators, the End-to-End Services Research Group, the authors of the above works, and many of our other colleagues have also been instrumental in clarifying our thoughts. In particular, Abhay Parekh has taught us much about the delay bound

results in [28, 29]. Also, Sally Floyd and Van Jacobson have rightly insisted that packet scheduling algorithms must deal with packet dropping and hierarchical link-sharing; we wish to acknowledge that much of our thinking on the hierarchical nature of link-sharing was stimulated by, and borrows heavily from, their work.

References

[1] S. Casner. *private communication*, 1992.

[2] D. Clark and V. Jacobson. *Flexible and Efficient Resource management for Datagram Networks*, unpublished draft, 1991.

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

[4] R. Chipalkatti, J. Kurose, and D. Towsley. *Scheduling Policies for Real-Time and Non-Real-Time Traffic in a Statistical Multiplexer*, in **Proceedings of GlobeCom '89**, pp 774-783, 1989.

[5] R. Cocchi, D. Estrin, S. Shenker, and L. Zhang. *A Study of Priority Pricing in Multiple Service Class Networks*, in **Proceedings of SIGCOMM '91**, pp 123-130, 1991.

[6] R. Cocchi, D. Estrin, S. Shenker, and L. Zhang. *Pricing in Computer Networks: Motivation, Formulation, and Example*, preprint, 1992.

[7] 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.

[8] J. DeTreville and D. Sincoskie. *A Distributed Experimental Communications System*, In **IEEE JSAC**, Vol. 1, No. 6, pp 1070-1075, December 1983.

[9] D. Ferrari. *Client Requirements for Real-Time Communication Services*, In **IEEE Communications Magazine**, 28(11), November 1990.

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

[11] D. Ferrari, A. Banerjea, and H. Zhang *Network Support for Multimedia*, preprint, 1992.

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

[13] S. Floyd. *Link-sharing and Resource Management Models for Packet Networks*, preprint, 1993.

[14] S. J. Golestani. *A Stop and Go Queueing Framework for Congestion Management*, In **Proceedings of SIGCOMM '90**, pp 8-18, 1990.

[15] S. J. Golestani. *Duration-Limited Statistical Multiplexing of Delay Sensitive Traffic in Packet Networks*, In **Proceedings of INFOCOM '91**, 1991.

[16] R. Guérin and L. Gün. *A Unified Approach to Bandwidth Allocation and Access Control in Fast Packet-Switched Networks*, In **Proceedings of INFOCOM '92**.

[17] R. Guérin, H. Ahmadi, and M. Naghshineh. *Equivalent Capacity and Its Application to Bandwidth Allocation in High-Speed Networks*, In **IEEE JSAC**, Vol. 9, No. 9, pp 968-981, September 1991.

[18] J. Kurose. *Open Issues and Challenges in Providing Quality of Service Guarantees in High-Speed Networks*, In **Computer Communication Review**, 23(1), pp 6-15, 1993.

[19] J. Hyman and A. Lazar. *MARS: The Magnet II Real-Time Scheduling Algorithm*, In **Proceedings of SIGCOMM '91**, pp 285-293, 1991.

[20] J. Hyman, A. Lazar, and G. Pacifici. *Real-Time Scheduling with Quality of Service Constraints*, In **IEEE JSAC**, Vol. 9, No. 9, pp 1052-1063, September 1991.

[21] J. Hyman, A. Lazar, and G. Pacifici. *Joint Scheduling and Admission Control for ATS-based Switching Nodes*, In **Proceedings of SIGCOMM '92**, 1992.

[22] 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.

[23] V. Jacobson and S. Floyd *private communication*, 1991.

[24] V. Jacobson *private communication*, 1991.

[25] S. Jamin, S. Shenker, L. Zhang, and D. Clark. *An Admission Control Algorithm for Predictive Real-Time Service*, In **Proceedings of the Third International Workshop on Networking and Operating System Support for Digital Audio and Video**, 1992.

[26] 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.

[27] R. Nagarajan and J. Kurose. *On Defining, Computing, and Guaranteeing Quality-of-Service in High-Speed Networks*, In **Proceedings of INFOCOM '92**, 1992.

[28] A. Parekh and R. Gallager. *A Generalized Processor Sharing Approach to Flow Control- The Single Node Case*, In **Technical Report LIDS-TR-2040**, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1991.

[29] 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.

[30] C. Partridge, *A Proposed Flow Specification* RFC-1363, July 1992.

[31] H. Schulzrinne *private communication*, 1992.

[32] H. Schulzrinne, J. Kurose, and D. Towsley. *Congestion Control for Real-Time Traffic*, In **Proceedings of INFOCOM '90**.

[33] S. Shenker *Service Models and Pricing Policies for an Integrated Services Internet*, to appear in **Proceedings of "Public Access to the Internet"**, Harvard University, 1993.

[34] S. Shenker, D. Clark, and L. Zhang. *A Scheduling Service Model and a Scheduling Architecture for an Integrated Services Packet Network* preprint, 1993.

[35] D. Verma, H. Zhang, and D. Ferrari. *Delay Jitter Control for Real-Time Communication in a Packet Switching Network*, In **Proceedings of TriCom '91**, pp 35-43, 1991.

[36] C. Weinstein and J. Forgie. *Experience with Speech Communication in Packet Networks*, In **IEEE JSAC**, Vol. 1, No. 6, pp 963-980, December 1983.

[37] D. Yates, J. Kurose, D. Towsley, and M. Hluchyj. *On Per-Session End-to-End Delay Distribution and the Call Admission Problem for Real Time Applications with QOS Requirements*, In **Proceedings of SIGCOMM '93**, to appear.

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

A On Standardizing a Service Model

Let us assume, for the sake of argument, that the Internet community agrees to adopt a service model similar in spirit to the one proposed here. There is then the question of how one *standardizes* the service model. There are two approaches. First, one could identify a single packet forwarding algorithm which supports this service model and then require that all routers use this algorithm. This entails standardizing the detailed packet scheduling mechanisms in the routers. It is not clear that all router technologies will be able to implement this particular packet scheduling mechanism, and so this approach may limit the range of technologies that can be employed in the Internet. One expects, in fact for the sake of progress one hopes, that the future Internet will have a diverse set of underlying technologies, and so requiring uniformity of the packet forwarding algorithm is probably not realistic nor desirable. The second approach involves adopting the service model without specifying the underlying mechanism. This path, while not nearly as straightforward (in fact, it poses a special challenge to the Internet's standardization procedures), is far more flexible. In this second approach there are several different conceptual issues which must be dealt with: (1) what services will be offered, (2) how are those services requested by the application, and (3) how are those services provided by the network. In this section we briefly address these three issues.

A.1 Defining the Services

There are two separate components to defining the set of services offered by the network: the service model and the service specification.

Service Model This is the basic set of services offered by the network and, as such, is the central expression of the network architecture. As we have argued previously, the service model should be based on fundamental application requirements. We have proposed a core service model in this memo. For individual flows it provides for two kinds of real-time service, guaranteed service and predictive service, along with multiple levels of elastic service. The service model also provides for hierarchical link-sharing services between collective entities.

Service Specification This is the detailed parameterization of the service model. This specification details how the service delivered to flows is characterized (e.g., delay bounds, etc.). In addition, the service specification details both how flows characterize themselves to the network (e.g., token bucket, leaky bucket, peak rate, etc.), and how these characterizations are enforced the network (e.g., by dropping or delaying packets, at entry points or at every router, etc.). While the service model is derived from rather general principles, the service specification involves making a number of detailed (and perhaps arbitrary) engineering choices.

A.2 Obtaining the Services

There are three kinds of services: link-sharing, elastic, and real-time. The link-sharing services will presumably be controlled through a network management interface. Since this network management interface will not typically be invoked by widely-used applications, there are few compatibility constraints. Thus, this interface can gradually evolve and so we need not be concerned with making its definition precise now. Since providing elastic service requires no preallocation of resources, we presume that applications utilizing elastic service will not need to pass through admission control. These elastic service desires (i.e., which level of ASAP service, and the preemptability of the packets) will probably be specified by the application in the interface to the transport protocol, and this will in turn be communicated to the network through some indication in the individual packet headers. We assume that this will be addressed in some other standardization venue, and we will not address it further here.

In contrast, providing the real-time services does require preallocation of network resources. Applications desiring real-time service will have to first explicitly request that service, and this request involves reserving network resources. The reservation procedure has two steps; first the application will invoke some operating system interface to request the reservation, and then some *set-up* or *reservation* protocol will forward that request to the network and return an answer. The application logically sees a request-response semantics through some operating system interface; the routers interact not with the application but with the reservation protocol and that can have a different interface. The set-up protocol has its own "service model" of the various configurations of reserved state it can provide; these were deemed *reservation styles* in [38] (we are not advocating the particular reservation styles in [38] but are merely citing them as examples of nontrivial relationships between flows and reserved resources). As an example of this we note that so far in our exploration of the service model, we have discussed only the service given to actual flows (that is, flows whose packets are forwarded by the network). However, one can reserve resources for potential flows as well; potential flows are those whose packets are not currently being forwarded, but for whom resources have been reserved.

Thus, in defining how applications obtain real-time services, we must deal with the reservation model of the set-up protocol and not just the service model of the network itself. We also must describe what information is passed between the applications and the network, and then must provide a detailed description of the interface invoked by applications. More specifically, the three conceptual pieces are:

Reservation Model The reservation model describes what configurations of resources can be reserved. The reservation model must not only address the service model available to individual flows, but must also incorporate more general configurations of reserved resources.

Negotiation Model The negotiation model describes, at an architectural level, (1) what parameters the application hands to the operating system interface, and (2) what parameters the application gets back from that interface. The negotiation model will depend on, and may therefore dictate, which end (source, receiver) of the application is submitting the requests. The negotiation model will also have implications for the set of queries and responses implemented in the admission control algorithm.

Reservation Interface This is a detailed parameterization (essentially the API) of the negotiation model. This reservation interface will be the artifact that is invoked by applications. It should be designed to be properly extensible, so that new services can be added, but it will inevitably be subject to compatibility constraints and therefore the previously defined components will be largely immutable.

A.3 Providing the Services

The previous two sections specify the range of services available, and how an application can obtain them. If there were a single network provider, then we would just require that the network support the interface to the set-up protocol and deliver the desired service. However, the Internet is, and will likely continue to be, a very heterogeneous and administratively decentralized institution. The service delivered to a flow is a concatenation of the service provided by the various routers along its path, and these routers need not implement the same packet forwarding algorithms. Thus, we need to directly address how we can be assured that such a network, with local operators making decisions about which routers to install, will indeed support the service model. As mentioned previously, one approach is to insist on a single router mechanism. The approach we advocate is, instead, to provide a functional requirement on routers rather than a definition of the detailed mechanism.

Router Interoperability Requirements This specifies a set of criteria that a router has to conform to. There are two categories of criteria. First, the routers must implement the interface used by the set-up protocol, and the admission control algorithm must support the appropriate set of queries. Incorporated within this is something functionally equivalent to what is described in RFC 1363 [30], which describes the set of parameters handed to routers along the path of a flow. Second, the service delivered by the router must conform to certain standards; these standards are designed so that the service delivered by a heterogeneous set of conforming routers will obey the service model. For guaranteed service, one might require that routers must approximate the WFQ *fluid model*, as defined in [7]. One can express the accuracy to which a router supports the fluid model with an error term which can be carried in the reservation interface as it is passed between routers and added up to compute the resulting end-to-end delay bounds. We should note that this is just one example; there are other alternatives for specifying how to deliver guaranteed service. For predictive service, the issue is much more difficult. Here the performance depends crucially on the admission control algorithm[15], and it is difficult to accurately characterize a measurement-based admission control algorithm. We propose that the performance of such algorithms be characterized by their performance on various test suites. These test suites will reveal not only the degree to which the delay bounds are met, but also the level of network utilization obtained (which depends on the behavior of the admission control algorithm). How one defines and evaluates such test suites is an unexplored yet crucial issue.

[15] The guaranteed service depends on admission control as well, but for guaranteed service there is a clear correctness condition for admission control. There is no such clear correctness condition for predictive admission control.