A Scheduling Service Model and a Scheduling Architecture for an Integrated Services Packet Network

Scott Shenker[1]
David D. Clark[2]
Lixia Zhang[1]

Abstract

Integrated Services Packet Networks (ISPN) are designed to integrate the network service requirements of a wide variety of computer-based applications. Some of these services are delivered primarily through the packet scheduling algorithms used in the network switches. This paper addresses two questions related to these scheduling algorithms. The first question is: what scheduling services should an ISPN offer? In answer, we propose a scheduling service model for ISPN's which is based on our projections about future application and institutional service requirements. Our service model includes both a delay-related component designed to meet the ergonomic requirements of individual applications, and also a hierarchical link-sharing component designed to meet the economic needs of resource sharing between different entities. The second question we address is: what implications does this service model have for the packet scheduling algorithms? We answer this question by constructing a scheduling architecture, and then argue that any scheduling algorithm capable of supporting our scheduling service model must conform to this architecture. The scheduling architecture is derived from the natural precedence ordering of the service model's various scheduling goals.

[1]Palo Alto Research Center, Xerox Corporation
[2]Laboratory for Computer Science, MIT.

1 Introduction

The current Internet, and most similar networks, 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 cannot efficiently deliver any other service model. This single class "best effort" service model provides the same quality of service to all flows[3]; 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 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. 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.

We expect that, in order to efficiently integrate the requirements of a wide variety of applications, the next generation of wide-area computer networks will offer a significantly more sophisticated service model. There is widespread consensus, in both the telephony and computer networking communities, that such networks should use packet-switching rather than circuit switching because packet-by-packet multiplexing uses bandwidth more efficiently than circuit-by-circuit multiplexing in the presence of bursty traffic. We will refer to packet-switched networks which support sophisticated service models as Integrated Services Packet Networks (ISPN).

One natural question is: what service model should an ISPN offer? This question is motivated by the design philosophy that 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. While both the underlying network technology and the overlying suite of applications will evolve, the need for compatibility requires that this service interface remain stable[4]. Thus, the

[3] *Flow* is the term we use to refer to end-to-end connections and other more general varieties of traffic streams.

[4] Actually, compatibility only demands that the existing parts of the service model must remain largely unchanged; however, the service model can be augmented 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; our service model will also have some components (link-sharing) which are exercised through a network management interface, and here the compatibility arguments do not apply with nearly the same force.

service model should not be designed in reference to any specific network artifact but rather should be based on fundamental service requirements. Because of its enduring impact, the choice of the service model is perhaps the single most important design decision in building an ISPN[5].

In order to efficiently support this more sophisticated service model, an ISPN must employ an equally 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. Once the networking community decides on an ISPN service model, a second natural question arises: which packet scheduling algorithms can realize this ISPN service model?

This paper discusses both the definition of an ISPN service model and also the interplay between the ISPN's service model and its packet scheduling algorithms. In the first part of our paper, we address the first question by proposing a subset of the service model, which we call the *scheduling service model*. The scheduling service model contains only those services that are related directly to the packet scheduling algorithm. We expect that the scheduling service model will form the core component of the full ISPN service model, and thus is deserving of special focus. We motivate our proposed scheduling service model by discussing the fundamental service requirements that an ISPN will need to meet. This detailed discussion of service requirements is one of the key novelties of our approach.

Since the scheduling service model focuses on the packet scheduling algorithm, there are many services that are not included in our scheduling service model. In particular, we exclude 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 scheduling of packets but which more fundamentally involve nonscheduling factors such as buffer management and inter-switch acknowledgment algorithms. Furthermore, we do not consider 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 we expect that many of these services will be offered by any future ISPN, they will not affect the basic scheduling service model and thus we do not expect that they will significantly affect the packet scheduling algorithms used in the internal switches.

In the second part of our paper, we address the second question by investigating what implications our scheduling service model has for packet scheduling algorithms. Recall that there is a tight coupling between the current Internet service model and the underlying FIFO scheduling algorithm. Similarly, we ask whether one can make any statements about the general structure, or architecture, of the packet scheduling algorithms that are needed to realize this ISPN scheduling service model. It turns out that, once one recognizes the natural precedence ordering between the various components of the scheduling service model, there is a canonical scheduling architecture dictated by our proposed ISPN scheduling service model. While there are many packet scheduling algorithms which realize the ISPN service model, we argue that they all must conform to the basic

[5]Reference [2], and to a lesser extent reference [15], also focus on flexibility of the packet scheduler as a primary design objective. We discuss this at greater length in Section 9.

architecture that we develop.

In this paper, we do not address the design of specific packet scheduling algorithms except to present briefly one particular instantiation of our architecture[6] which demonstrates that our service model is not impractical. We should note that there have been many other packet scheduling algorithms proposed in the literature (see, for example, [12, 14, 16, 17, 25, 28, 30, 31, 35]), and they too implement various pieces of our service model.

A packet scheduling algorithm is only part of a complete mechanism to support explicit qualities of service. In particular, since resources are finite, one cannot 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. While in this paper we focus on the scheduling service model and on the architecture of scheduling algorithms, 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 [14, 19, 23, 27] for 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 work is a revised version of the first half of Reference [3], which contains an embryonic form of the thinking presented here. However, we would like to acknowledge that the thoughts discussed in this paper also reflect the contributions of many others. In particular, the works of Parekh and Gallager [30, 31], Ferrari et al. [12, 14, 35], Jacobson and Floyd [2, 25, 15], Golestani [16, 17], Guerin et al. [18, 19], Kurose et al. [4, 20, 29, 33, 37], Lazar et al. [21, 22, 23, 24], and Kalmanek et al. [28] have been critical in shaping our thinking on this matter. Discussions with 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 [30, 31]. 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.

This paper has 10 sections. In Section 2 we identify the two kinds of quantitative 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 present a precedence ordering among these service commitments and then in Section 6 we argue that this ordering leads to a particular packet scheduling architecture. In Section 7 we present an instantiation of this architecture. In Section 8, we review the various forms denial of service can manifest, and the ways in which denial of service can be used to augment the scheduling service model. We review the related literature in Section

[6]A fuller description of this packet scheduling algorithm will be forthcoming in a revision of the mechanism presented in the second half of Reference [3].

9, and then conclude in Section 10.

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 scheduling 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 scheduling 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 (at this time, certainly greater demand than for the quality of service commitments to individual flows); 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 scheduling service model.

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 future high-speed networks), the time of delivery is the only quantity of interest to applications. Delay is clearly the most central quality of service, and thus 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 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 [29] 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

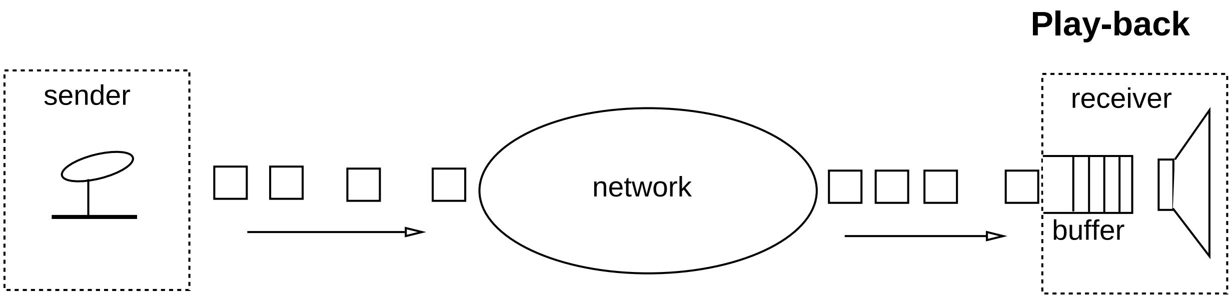

Figure 1: A schematic diagram of a playback application. The signal is generated and packetized at the sender and then transmitted over the network. The receiver, in order to remove the effects of network-induced delay jitter, buffers the packets until their playback points.

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 consider the delay requirements of these two classes separately.

3.2.1 Real-Time Applications

An important class of such real-time applications, which is 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[7].

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 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 [8]. 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

[7] 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.

[8] 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.

(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. 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 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 should adapt 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.

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 [9, 36]); the VT [1] and VAT [26] 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.

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, 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[9] 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

[9]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.

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 a 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; of course, the trivial *no-queueing* minimum 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[10]. 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[11].

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 [12]. 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

[10]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.

[11]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.

[12]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.

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 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[13]; 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. A fundamental point of our overall architecture is that traffic characterization and admission control are necessary for these real-time delay bound services.

The third 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[14]. Elastic applications vary greatly in their sensitivity to delay (which, as we mentioned earlier, is probably more a function of the average delay than of the maximum delay), and so the service

[13]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, 32].

[14]We choose not to use the term "best-effort" for the ASAP service since that connotes the FIFO service discipline. Also, we should note that we do not describe, as part of the scheduling service model, any congestion control related feedback (congestion notification bits, etc.) which might be part of such a 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.

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-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

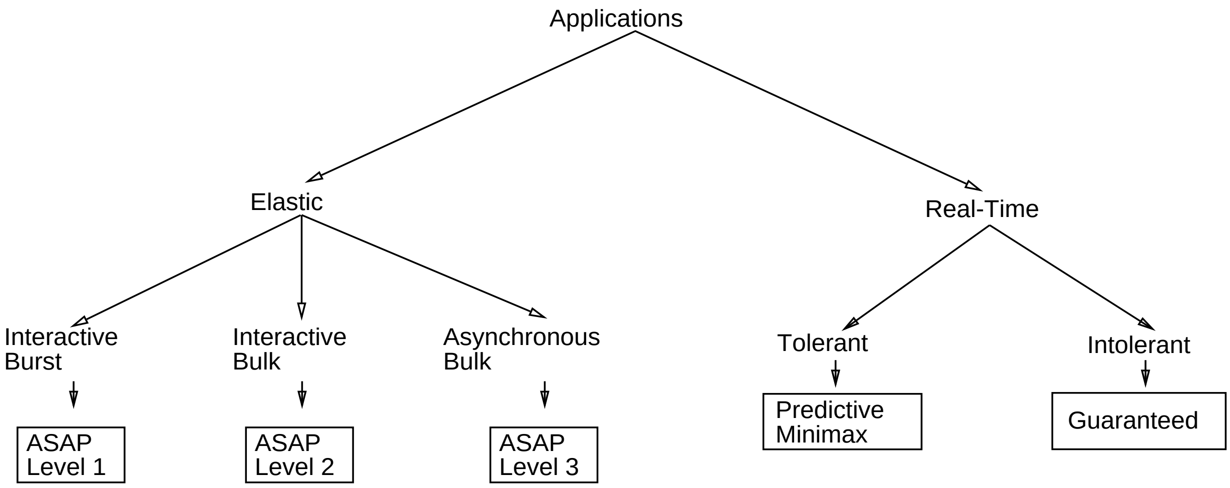

Figure 2: Our rough taxonomy of applications and their associated service models. We have arbitrarily depicted three levels of ASAP service.

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 scheduling service model. The resulting scheduling service model should be judged not on the validity of the underlying taxonomy but rather on its ability to adequately meet the needs of the entire spectrum of applications. In particular, not all real-time applications are playback applications; for example, one might imagine a visualization application which merely displayed the image encoded in each packet whenever it arrived. However, non-playback applications can still use either the guaranteed or predictive real-time service model, although these services are not specifically tailored to their needs. Similarly, playback applications cannot be neatly classified as either tolerant or intolerant, but rather fall along a continuum; offering both guaranteed and predictive service allows applications to make their own tradeoff between fidelity 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 scheduling service model can adequately meet all application needs.

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 questions 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 [8] and further explored in [30, 31]) 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[15] 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

[15]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 $c_i' > 0$).

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 [25]. 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, 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

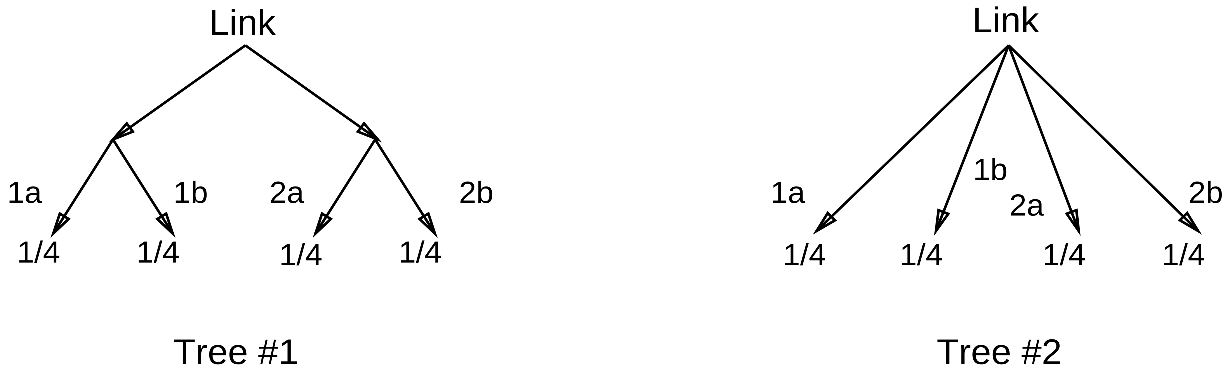

Figure 3: Two possible sharing trees with equal shares at all leaf nodes. When one of the leaf nodes is not active, the trees produce different bandwidth shares for the remaining active nodes.

via quality of service commitments and not through the link-sharing service model. Therefore, for our resource-sharing service model we restrict ourselves to the hierarchical service model presented above.

This preceding discussion about the link-sharing service model implicitly assumed that all traffic associated with a bottom-level entity was serviced in a FIFO manner (i.e., the packet's belonging to a bottom-level entity were never reordered, even though the ordering relative to packets from other bottom-level entities was not necessarily FIFO). However, the link-sharing service model will need to coexist with other scheduling disciplines. Rather than defining a completely general service model, we will later present an example where link-sharing is defined in the context of non-FIFO scheduling.

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 Ordering the Service Requirements

Our collection of service models consists of guaranteed real-time, predictive real-time, several classes of ASAP, and hierarchical link-sharing. These service models are comprised of several varieties of service commitments. There are three different quantitative service commitments: guaranteed delay bounds, predictive delay bounds, and link-sharing bandwidth allocation shares. There are also two different relative service commitments: minimax for predictive traffic and multiple classes of ASAP for elastic traffic.

These service commitments can be seen as a set of objectives or goals for the packet scheduling algorithm. However, when a packet arrives and is scheduled, all of these objectives must be combined in some manner to make a single consistent scheduling decision for the packet. This is not an entirely trivial task, given that there can be conflicts among the objectives. For example, a packet using guaranteed service may need to leave at once to meet its delay objectives, but may exceed the link-sharing objective if it does so. Furthermore, in a more trivial way, every other scheduling objective is in conflict with the ASAP scheduling goals. We must therefore find a way of systematically combining these various objectives into a coherent decision framework.

We now define a transitive precedence ordering among the scheduling goals associated with these service commitments. This ordering can be thought of as defining a decision tree that reflects how the various objectives should be used to make a single scheduling decision. As such, this precedence ordering reflects which service commitments the packet scheduling algorithm should keep if, upon overload, it cannot meet all of them and thus must choose between them. Two important points to keep in mind are that (1) this is *not* an ordering of importance of the various service objectives but instead an ordering of which criteria take precedence in the scheduling algorithm, and (2) this ordering is not done in isolation but rather takes into account the limitations imposed by admission control.

Consider the three quantitative service commitments: guaranteed delay bounds, predictive delay bounds, and link-sharing bandwidth allocation shares. Recall that the delay bound given to guaranteed real-time traffic is advertised to be perfectly reliable, whereas the delay bound given to predictive real-time traffic is explicitly advertised to be imperfectly reliable. This strongly suggests that guaranteed delay bounds take precedence over predictive delay bounds. In addition, since the link-sharing service model is less concerned with the timing of each individual packet than the real-time service models, we can therefore conclude that the guaranteed delay bounds and the predictive delay bounds take precedence over the link-sharing bandwidth allocation shares on a packet by packet basis. Admission control plays a significant role here. One can only allow the real-time delay bound scheduling goals to take precedence over the link-sharing scheduling goals because admission control ensures that such a policy will not lead to a significant violation of the link-sharing goals.

In general, it is natural to give quantitative service commitments precedence over qualitative ones. Correspondingly, we give the link-sharing scheduling goals, and therefore by transitivity the real-time delay bound scheduling goals, precedence over the ASAP scheduling goals. Furthermore, we give the real-time delay bound scheduling goals precedence over the minimax scheduling goals. However, we do not give the link-sharing scheduling goals precedence over the minimax scheduling goals, for the following reason. Admission control must ensure that the real-time traffic, by itself, does not lead to violations of the link-sharing bandwidth allocation shares. This means that we do not have to check link-sharing limits when we make scheduling decisions for individual real-time packets. Consequently, we need not give the link-sharing scheduling goals precedence over the minimax scheduling goals.

Note, however, that we did insist that the real-time delay bound goals took precedence over the minimax scheduling goals; this is because we do not believe that admission control alone can ensure that real-time delay bounds will be met[16].

[16]This statement is in contrast to the assumption made in much of the ATM literature that admission control is not only necessary but also sufficient to ensure real-time delay bounds. We, to the contrary, do not expect that networks will be able to support real-time delay bounds while operating at reasonably high levels of utilization without explicit help from the packet scheduling algorithm.

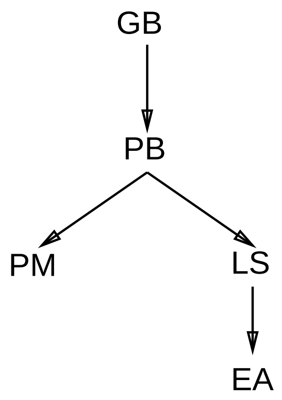

Figure 4: The precedence ordering of the various scheduling goals. An arrow indicates precedence and the following acronyms are used: GB=guaranteed delay bound, PB=predictive delay bound, PM=predictive minimax, LS=link-sharing bandwidth allocation share, and EA=elastic ASAP.

We have thus established the ordering relationships of the three quantitative service commitments. The guaranteed delay bounds takes precedence over everything else, the predictive delay bounds take precedence over everything except the guaranteed delay bounds, and the link-sharing bandwidth shares take precedence over the ASAP scheduling goals but do not take precedence over the minimax scheduling goal for predictive traffic. We now claim that the two remaining relative service commitments, minimax predictive and ASAP elastic, are not directly comparable. While currently it seems clear that the time scales of the service requirements for typical elastic applications are larger than those for typical tolerant real-time applications, there is nothing in the distinction between elastic applications and tolerant real-time applications that demands that this remain so in the future. We do not wish to embed this perhaps temporary time-scale distinction in our architecture, and thus do not declare a precedence ordering relation between these two relative scheduling goals. Consequently, since we are assuming this is a transitive ordering, we cannot install a precedence ordering between the relative service commitment of minimax predictive and the quantitative service commitment of link-sharing bandwidth allocation shares; above we argued that it was not necessary to give precedence to link-sharing over minimax predictive, and now due to transitivity we find that we cannot give precedence to minimax predictive over link-sharing because that would then imply a precedence ordering between minimax predictive and ASAP elastic. Combining these precedence relations, we find the precedence ordering of the scheduling goals which is depicted in Figure 4.

One subtle point that arises is the interaction between the scheduling goals of guaranteed traffic and the qualitative goals of minimax for predictive traffic and ASAP for elastic traffic. Recall that guaranteed service makes a firm commitment that packets will arrive before the delay bound, but makes absolutely no commitments about when during this period the packets will arrive. However, the two other services, elastic and predictive, do make qualitative commitments about decreasing delay. In order for guaranteed service to be compatible with these other commitments, guaranteed packets should never take precedence over other packets unless they must be sent in order to realize the delay bounds, and guaranteed packets should always be sent if the link would otherwise be idle. Thus, guaranteed packets should be sent as late as possible without violating

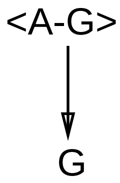

Figure 5: A scheduling architecture for guaranteed service. The symbol G represents some algorithm that orders the guaranteed packets and $< A - G >$ represents an arbiter which decides when to send guaranteed packets.

the delay bounds or letting the link go idle.

When seen as defining a decision tree, this precedence ordering sets up a sequence of tests by which scheduling decisions are made. First, the algorithm must check if any guaranteed packets need to be sent in order to satisfy a guaranteed delay bound; if so, they are sent. Second, the algorithm must check if any predictive packets need to be sent in order to satisfy a predictive delay bound; if so, they are sent. If no real-time packets need to be sent in order to satisfy these real-time delay bounds, then the algorithm is free to arbitrarily choose between giving some predictive packet service (thereby meeting a minimax service goal), or giving some elastic packet service (thereby meeting an ASAP service goal). When the algorithm chooses to service an elastic packet, the link-sharing goals determine the which elastic packet is sent.

6 A Scheduling Architecture

In the last section we argued that general properties of the service model led to a natural precedence ordering of the various service commitments. We now explore the implications this precedence ordering has for packet scheduling algorithms. One might think that the precedence ordering of scheduling goals would lead to a strict priority scheduling algorithm. We now argue that, instead, this ordering leads to a rather different architecture for packet scheduling. For the moment, we will ignore the bandwidth related scheduling goals associated with link-sharing and concentrate on the other scheduling goals which are all delay related. We first consider the case when there are only guaranteed service commitments, and then add predictive service commitments and finally elastic service commitments. We can represent a general architecture for scheduling algorithms which deliver guaranteed service with the diagram in Figure 5. In this Figure, G represents some algorithm that orders the guaranteed packets and $< A - G >$ is an arbiter which decides when to send the packet at the front of the queue. As we discussed in the previous section, $< A - G >$ is designed to send packets as late as possible without violating the bounds, but will send packets if the link would otherwise be idle.

When we add predictive service, the precedence ordering dictates that we get the structure depicted in Figure 6. In this Figure, P represents some algorithm that orders the predictive packets and the arbiter $< A - G >$ takes a packet from P unless it is necessary to service G to meet the guaranteed delay bounds. If both the predictive and guaranteed queues need to be serviced in

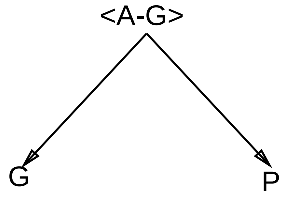

Figure 6: A scheduling architecture for guaranteed and predictive service. The symbol P represents some algorithm that orders the predictive packets and $< A{-}G >$ represents an arbiter that decides which queue to serve. The arbiter services the G queue only when necessary to meet the guaranteed delay bounds, but if both queues need servicing the G queue gets precedence.

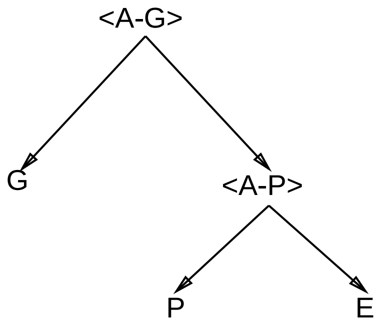

Figure 7: A scheduling architecture for guaranteed, predictive, and elastic service. The symbol E represents some algorithm that orders the elastic packets, and $< A - P >$ represents an arbiter that decides which queue to serve. This arbiter must meet the predictive delay bounds, but is otherwise arbitrary.

order to prevent a violation of their bounds, the guaranteed queue takes precedence.

The scheduling requirements of meeting the delay bounds of real-time traffic takes precedence over the ASAP scheduling requirements of elastic traffic. However, the scheduling goals of minimax for predictive traffic and ASAP for elastic traffic do not take precedence over each other. This leads to the structure illustrated in Figure 7, where $< A - P >$ is an arbiter which makes sure that the delay bounds of predictive real-time traffic are met but which then allocates service between the ASAP needs of the elastic traffic and the minimax needs of the predictive traffic. This nature of this allocation is not specified by the architecture, and can be anything consistent with the predictive delay bounds.

We now add link-sharing to this structure. Since link-sharing comes after the predictive bounds but before the elastic ASAP in Figure 4, we get the structure shown in Figure 8. Here the E_i denote algorithms that order the elastic traffic belonging to the various bottom-level entities and $< LS >$ refers to a link-sharing algorithm that approximates the ideal fluid model; the link-sharing hierarchy is embedded within the link-sharing algorithm $< LS >$, so we do not explicitly show the whole link-sharing tree. The key point here, which is not explicitly shown in the above diagram, is that the link-sharing algorithm must take into account the bandwidth used by the guaranteed and predictive packets sent by the various collective entities when deciding which elastic traffic to send, although the link-sharing algorithm does not affect the scheduling of the guaranteed and predictive

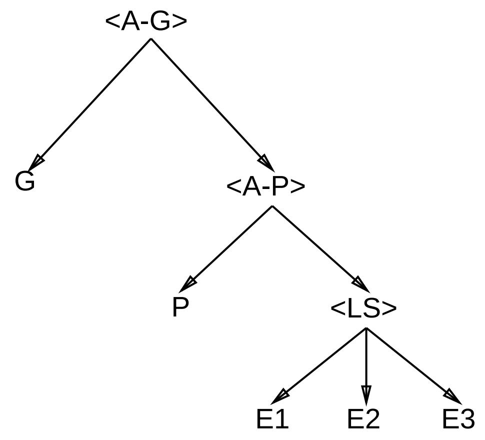

Figure 8: A scheduling architecture for guaranteed, predictive, elastic, and link-sharing service. The symbols E_i represent algorithms that orders the elastic packets belonging to various bottom-level entities, and $< LS >$ represents the link-sharing algorithm.

packets. To make this more precise, recall that in the ideal service model of link-sharing each entity i is represented by a node in the link-sharing tree that has associated with it a virtual queue, a share of the parent node's bandwidth, and the quantities $c_i(t)$, which represents the arrival pattern of bits, and $b_i(t)$, which represents the number of bits remaining in the virtual queue. Previously, when we considered link-sharing in the absence of quality of service commitments, each node's virtual queue was FIFO; to take account of the real-time traffic we modify this. Whenever an elastic packet belonging to entity i or one of its descendants arrives, we place the corresponding idealized packet in the rear of the node's virtual queue. Whenever a real-time packet belonging to entity i or one of its descendants is transmitted, the corresponding idealized packet is placed at the front of the virtual queue. Thus, the transmission of a real-time packet belonging to entity i has the effect of delaying the departure of the queued elastic packets in the idealized model. The cumulative bandwidth of the elastic and real-time traffic belonging to entity i will therefore match the desired policy requirements.

Our examination of the scheduling service model, and the precedence ordering of the scheduling goals contained therein, leads us to conclude that any packet scheduling algorithm which supports our scheduling service model will conform to the architecture depicted in Figure 8. This architecture has three notable pieces: the guaranteed scheduling algorithm, which is comprised of the arbiter $< A - G >$ and the ordering algorithm G; the predictive scheduling algorithm, which is comprised of the arbiter $< A - P >$ and the ordering algorithm P; and the link-sharing algorithm $< LS >$ and the ordering algorithms E_i. The scheduling architecture details how these pieces fit together. Guaranteed service sits at the top of the structure, followed by predictive service. Link-sharing goes below both of these services. Thus, our architecture dictates that while the link-sharing goals will affect the admission control decisions for real-time flows, the link-sharing goals have no effect on the scheduling of the real-time packets and only affect the scheduling of elastic packets. We maintain that this is not just one possible way of scheduling packets, but rather the *only* way consistent with our service model.

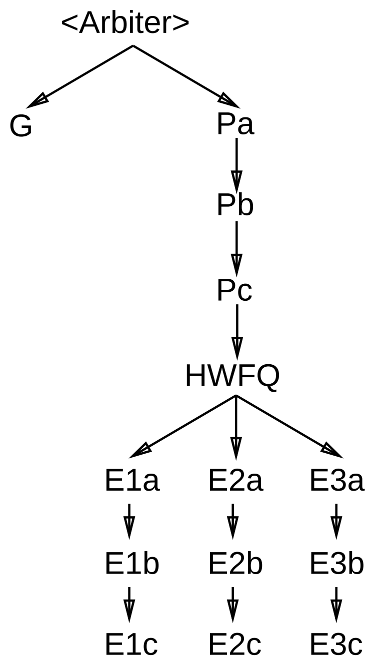

Figure 9: A schematic diagram of a packet scheduling algorithm which realized the proposed scheduling service model. The labels a, b, c indicate different priority levels in the predictive and elastic traffic classes. The G queue is ordered by the SVC timestamps, and the P and E queues are all FIFO. The arbiter serves the G queue only when the lead timestamp is greater than current time.

7 An Instantiation of the Scheduling Architecture

An instantiation of this architecture is defined by a choice for the arbitrating algorithms $< A-G >$ and $< A-P >$, the link-sharing algorithm $< LS >$, and the ordering algorithms G, P, and E_i. The combination of $< A-G >$ and G must provide perfectly reliable bounds, which means that the service a guaranteed flow gets must not be greatly affected by the behavior of other traffic flows. Thus, as we argue in [3], the key heuristic to keep in mind when designing these algorithms is *isolation*; the scheduling algorithm must isolate the flows from one another. There are many choices for $< A-G >$ and G: for instance, the "stop-and-go" algorithm in [16, 17], the hierarchical round-robin in [28], the J-EDD and D-EDD schemes in [12, 14, 35], the weighted round-robin algorithm described in [2, 25], and the weighted fair queueing (WFQ) algorithm described in [8] and later analyzed in [30, 31]. However, we will choose to use a "stalled" version of the VirtualClock [38, 39] algorithm which we will denote SVC and will describe in a later note but is essentially using VirtualClock timestamps to order the guaranteed packets and then only sending packets when their timestamp value is less than or equal to real time (or unless the link would otherwise be idle). While all of these algorithms provide guaranteed service, they vary in the degree to which they delay guaranteed packets until it is absolutely necessary to send them and they also vary in the efficiency with which they deliver guaranteed service (i.e., for a given amount of bandwidth, how many service commitments can be simultaneously met).

The predictive service model is to provide reliable bounds but to also deliver minimax service (that is, minimize the ex post maximum delay). Since we need not provide a perfectly reliable bound, isolation is not the most important requirement. In fact, isolation is counterproductive for predictive traffic and, as we argued in [3], the key heuristic here is *sharing*; sharing enables a particular flow's transient burst of traffic to pass through a switch without those packets experiencing overly large delays by spreading the delay around to other flows. Thus, an appropriate scheduling discipline is FIFO. (Actually, as we argue in [3], one can extend this notion of sharing across switches and then an appropriate scheduling algorithm is what we called FIFO+.) Since we may want to offer several different delay bounds, we will employ a multi-level strict priority queue.

The service model for link-sharing revolved around an idealized fluid model. The connection between such fluid models and actual scheduling disciplines is discussed in [8] and [30, 31]; suffice it here to say that this connection is usually done by assigning, in the real switch, a "timestamp" to each real packet based on when all the bits in the corresponding idealized packet have been transmitted in the fluid model, and then using these timestamps to order the packet transmissions. This straightforward realization of the fluid processor sharing model produces the WFQ scheduling algorithm (its use for link-sharing was first explored in [7]). The WFQ algorithm can be extended to a hierarchical WFQ algorithm to match the service model of hierarchical link-sharing. However, we assume that one could modify several other algorithms, such as weighted round-robin or VirtualClock, to provide reasonable approximations to this service model.

In order to meet the delay-related relative service commitments for elastic traffic, we can provide several levels in a strict priority queue[17]. Lastly, for the arbiter $< A - P >$ we choose to give strict priority to the predictive traffic. We do this because, as we mentioned previously, currently most real-time applications have tighter delay requirements than most elastic applications; this is accentuated by the fact that tolerant real-time applications are sensitive to the tail of the delay distribution whereas the performance of elastic applications tends to depend more on the center of the distribution.

Thus, a possible instantiation of the architecture is shown in Figure 9, where the labels a, b, c indicate different priority levels in the predictive and elastic traffic classes. In a later paper we will discuss the implementation and the performance this scheduling algorithm.

8 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.

[17]This does not address any of the congestion control issues that arise with elastic traffic. Congestion control may necesitate using some variant of Fair Queueing [8] along with some form of congestion feedback.

Since this paper is primarily about scheduling service models, and consequently focuses on the service actually delivered by the network as opposed to the service denied by the network, we do not address in detail the algorithms used to deny service. Instead, in this section we merely discuss the various kinds of denial of service and sketch a few ways in which denial of service can be used in conjunction with our service model. In particular, denial of service can be used to augment the resource sharing portion of the scheduling 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.

8.1 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 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 [14, 19, 23, 27] 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.

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 scheduling service model. We mention utilization targets here because other aspects of the scheduling 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.

8.2 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 [33]. 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 a *filler* for the network; they will be serviced only if they do not interfere with any other scheduling goal 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.

This paper is based on the assumption that one can usefully distinguish between packet scheduling decisions ("*which* packet do we send next?") and packet dropping decisions ("*if* this packet is next to be sent, should we send it or drop it?"). Such a distinction seems natural when dropping is a fairly rare event, and not the main vehicle through which quality of service is delivered. As we discussed in Section 5, packet scheduling decisions have an ordered structure, which we depicted in Figure 4. In contrast, decisions about when to drop packets may involve the entire suite of scheduling goals; one might drop expendable guaranteed packets in order to reduce ASAP delays, and one might drop predictive expendable packets in order to service additional expendable guaranteed packets. This complicated interrelationship of service goals and possible dropping decisions makes it difficult to envision a coherent and systematic architecture that describes which packets to drop and when[18]. Does this render our basic scheduling architecture invalid or irrelevant? We discuss this question below.

In this paper, we have discussed a service model in which we implicitly assumed that the vast majority of sent data is delivered to its destination. In our taxonomy, some applications needed explicit assurances about the network delays (the real-time applications), and others needed no such assurances (the elastic applications). However, the universal assumption was that most

[18]Reference [15] discusses using link-sharing ideas to control such dropping decisions; while this is one possible approach, more general approaches are also possible.

applications expected that all (or almost all) of their data would be delivered. Thus, the network was faced with a bursty and inflexible (inflexible in that most of the data was neither preemptable nor expendable) load, and the challenge was to deliver the desired qualities of service. Scheduling algorithms (accompanied by the appropriate admission control algorithms) are indeed the only fully general way to cope with this problem, and that was our object of focus.

One can imagine applications such as hierarchically-encoded video which could easily be adequately served with a service model in which, as we have briefly outlined in this section, there is no assumption that most sent data will be delivered. If these applications represent a small percentage of the traffic in the network, then such preemptable or expendable traffic can be seen as cushions or filler which merely ease the implementation of our original service model, but do not undermine its relevance for the other applications. However, if such applications become the dominant source of traffic in the network, then the central design problem is quite different. The network is now faced with a variable but highly flexible demand, and can therefore drop packets at will to ensure an almost constant delivered load. In this scenario, packet dropping, rather than packet scheduling, will be the main vehicle through which qualities of service are delivered and thus while the service model will remain valid, the scheduling architecture will be rendered largely irrelevant. We conjecture, and we admit that it is completely conjecture at this point, that the network will not become dominated with such expendable traffic. This is largely an economics judgment; if the network did become dominated with such expendable traffic, then the marginal cost of serving nonexpendable packets would be almost identical with the marginal cost of serving expendable packets (since there is already such a large pool of droppable packets), and so the two services should be similarly priced. However, one can only hypothesize a sizable share of expendable packets if expendable service has a significant price advantage over nonexpendable service. Thus, we think it unlikely that the network will be dominated by such expendable traffic.

9 Related Work

There has been a flurry of recent work on providing various qualities of service in packet networks. We cannot hope, nor do we try, to cover all of the relevant literature in this brief review. Instead, we mention only a few representative references. Furthermore, we focus exclusively on the issue of the service model and do not discuss to any great extent the underlying scheduling algorithm (for a review of the scheduling algorithms, see [3]).

The motivating principle of this work is that the service model is primary. However, Reference [2] (and, to a lesser extent, Reference [15]) 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 network. There are two points to be made.

First, 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 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. 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.

Second, 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 looking for 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. Our disagreement 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). Network architectures designed around such implicit quality of service mechanisms do not require a well-defined service model nor do they require charging for network service; the network architecture we have advocated involves explicit quality of service mechanisms and therefore requires a stable service model and, as we argue in Section 10, differential charges for the various levels of network service.

Much of the recent quality of service literature concentrates on the support of real-time applications. As is most clearly spelled out in References [11, 13], the consensus of the literature is that the appropriate service model for these real-time applications is to provide *a priori* delay bounds. 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 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. In this view, it would be sufficient to provide only the several classes of elastic service without *any* real-time services. We think that the extreme variability of the offered load will require too great a degree of overprovisioning to make this approach practical. However, our line of reasoning is well outside the scope of this paper; we hope to explore this in more detail in future work.

There are several service schemes whose service model is to provide a bound on the maximum delay of packets, provided that the application's traffic load conforms to some prearranged filter. Such schemes include the WFQ (see [8]; also see citeP-G,ap-thesis which refers to this as the PGPS algorithm), Delay-EDD (see [14]), and Hierarchical Round Robin (see [28]). This service model is identical to our guaranteed service model; this can be considered the canonical service model for supporting real-time applications.

There are several service schemes which, given that the application's traffic load conforms to some prearranged filter, 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. One such scheme is the Jitter-EDD scheme (see [11, 35]). The original Stop-and-Go scheme (see [16]) provides a jitter bound, which is a universal bound on the difference between the maximum and minimum delays which applies to all flows, no matter what network path their traffic takes, and no matter what their offered load is (as long as it conforms to the characterization handed to admission control). The maximum delay bound will depend on the path, but the jitter bound depends only on the frame size of the network which is fixed. Subsequent enhancements to this scheme (see [17]) enable the network to provide several different values of jitter bounds. 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, as we argued in Section 3.2, 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 scheduling service model.

A rather different form of service model is the offering of statistical characterizations of performance. The Statistical-EDD scheme (see [14]) offers a delay bound and the probability that bound will be violated. In the MARS scheme, delay bounds are firm but there is a statistical characterization of packet loss (see [21, 22]). In some ways, these service offerings are similar to the predictive delay bounds included in our service model; however, we do not supply a precise estimate of the probability. In fact, we explicitly rejected such statistically characterized service offerings (in Section 3.1) 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. The SMDS service interface (see [10]) offers a fixed delay bound (independent of path) with an assurance that a given percentage of the traffic will meet that bound. The statistical characterization offered here is more similar to our predictive service, in that it applies only over long time intervals. Another scheme which attempts to provide a reliable bound, but does not give a precise estimate of the probability of violation, is implicitly defined by the equivalent capacity approximations in References [18, 19]; these approximations, when used in an admission control scheme, can ensure with high reliability that delay bounds are not violated.

The link-sharing service model has been informally discussed for years, but has rarely been written about. One exception is the work of Davin and Heybey (see [7]), where an approximation to the WFQ algorithm was used to share a link between several agencies. More recently, Jacobson and Floyd [15, 25] have discussed the possibility of hierarchical link-sharing, and have proposed a mechanism to accomplish this. Steenstrup [34] has also proposed a mechanism for such hierarchical sharing. In most of these works [2, 7, 25], the service model has been implicitly defined by the mechanism itself. Recently, Floyd [15] has provided a more principled description of the service model, independent of the implementing mechanism. This service model is, in general outline, somewhat similar to what we have proposed. The biggest difference is that the service model is defined relative to *estimators*, which calculate an entity's bandwidth usage over some time period, and *persistent backlogs*, which indicate unsatisfied demand; in contrast, our service model is defined relative to the fluid model. It is not clear how these approaches differ in practice.

The idea of offering several classes of service to elastic traffic is often not explicitly mentioned in many of the above proposals, but represents an entirely trivial change to the various schemes.

This brief review of related work reveals that each component of our scheduling service model has some similar counterpart in the literature. While our service model is unique in including all of these different components, our service model does not contain the sum of the features of the aforementioned schemes. In particular, we have excluded nontrivial bounds on minimum packet delays and also excluded any statistically characterized service offerings.

Few of the above works focus on the service model independently of a particular realization. Consequently, they have typically not addressed the issue of the existence of a general scheduling architecture that we have proposed here. The only exception to this is the recent work of Floyd [15]. We have argued for a precedence ordering between the various scheduling goals, with real-time objectives taking precedence over link-sharing objectives. In contrast, Floyd views link-sharing as coequal with real-time objectives, and argues that in some cases the link-sharing goals should cause real-time bounds to be violated. This is a rather fundamental difference, with roots in the differing roles the two viewpoints ascribe to service commitments; we see quality of service as negotiated on a flow-by-flow basis with link-sharing only used for resource sharing issues, whereas Floyd sees link-sharing as another way in which to deliver quality of service to flows (which then renders it comparable in precedence to real-time goals). We hope to more fully explore the differences between these two viewpoints in future work.

10 Discussion

Figure 10 depicts our line of reasoning in this paper. In the first part of this paper, we proposed a scheduling service model for ISPN's. This proposal was based on some assumptions about the nature of present and future application quality of service requirements and institutional resource sharing needs. The proposal was also shaped by judgments about the practical limitations of what can be controlled through scheduling algorithms, and judgments about the relative efficiency with which the services can be delivered. Our service model has two components; a delay-related component designed to meet the ergonomic requirements of individual applications, and a link-sharing component designed to meet the economic needs of resource sharing between different entities. The delay related services include two kinds of real-time service, guaranteed service and predictive service, and also includes multiple classes of ASAP elastic service. The service model for hierarchical link-sharing is based on a hierarchical version of a fluid model for generalized processor sharing.

In the second part of this paper, we explored the family of scheduling algorithms that could realize

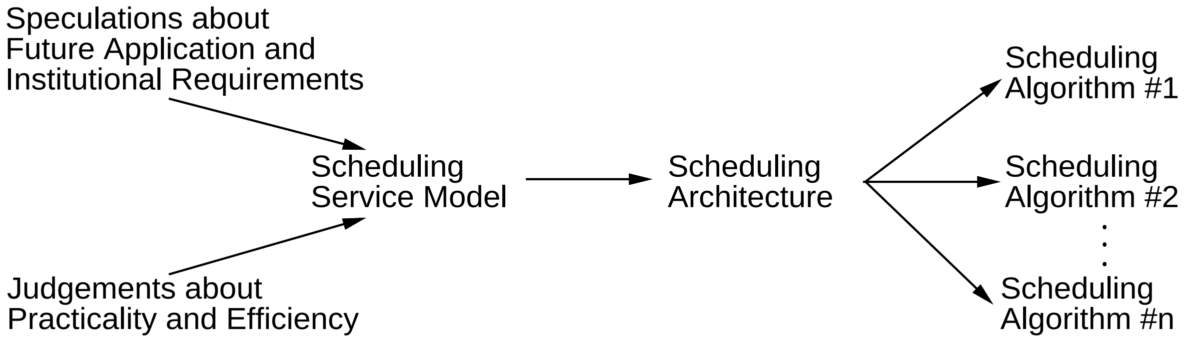

Figure 10: A schematic diagram of the line of reasoning used in this paper.

this scheduling service model. We first introduced a transitive precedence ordering of these service commitments. We then found that this ordering led directly to a canonical scheduling architecture; all scheduling algorithms supporting our service model must conform to this architecture. This architecture is fairly general and can have many different instantiations (and in Section 7 we sketched one such instantiation). The key elements of this architecture are that (1) predictive and elastic packets are sent if and only if guaranteed packets do not need to be sent, (2) admission control is used to keep the link-sharing goals from being violated by real-time traffic, and (3) the link-sharing algorithm accounts for the real-time traffic in scheduling the elastic traffic, but does not affect the scheduling of the real-time traffic.

The service model should be based on fundamental service requirements. Since we obviously don't know what future application requirements will be, the design of a scheduling service model is inherently a speculative task, and there will be legitimate disagreements about which service models are most appropriate. It is important to distinguish between disagreements which arise from different predictions about future applications and those disagreements which arise from different judgments about how to best serve an agreed upon set of applications. Therefore, it is crucial to make the assumptions about future applications explicit, and we have attempted to do that in this paper.

Despite the surfeit of detailed scheduling proposals, there has been a regrettable dearth of discussion, much less debate, about the basic service models that best fit application needs and network technology ([11, 13] are a notable exceptions). In addition, comparisons between various scheduling proposals typically focus solely on the algorithmic details rather than stressing the underlying architectural and structural aspects. Thus, while we obviously hope that the specific technical proposals contained in this note have some validity, we think it likely that the issues we address, those of service models and scheduling architectures, are more important than the specific answers we propose. In fact, perhaps the most important point we make is that the arrows in Figure 10 go from left-to-right rather than from right-to-left as is implicitly assumed in the more mechanistically based discussions of integrated services packet networks.

We conclude with one last observation: pricing must be a basic part of any complete ISPN architecture. If all services are free, there is no incentive to request less than the best service the network can provide, which will not produce effective utilization of the network's resources (see Reference [5, 6, 32] for a discussion of these issues). The sharing model in existing datagram networks deals with overload by giving everyone equally poor service; the equivalent in real-time services would be to refuse a high fraction of requests, which would be very unsatisfactory. Prices must be introduced so that some clients will request lower quality service because of its lower cost. Therefore, real-time services must be deployed along with some means for accounting.

It is exactly this price discrimination that will make the predictive service class viable. Certainly predictive service is less reliable than guaranteed service and, in the absence of any other incentive, network clients would insist on guaranteed service and the network would operate at low levels of utilization and, presumably, high prices. However, if one can ensure that the reliability of predictive service is sufficiently high and the price sufficiently low, many network clients will prefer to use the predictive service. This will allow ISPN's to operate at a much higher level of utilization, which then allows the costs to be spread among a much larger user population.

11 Acknowledgments

We would like to thank Sally Floyd, Steve Deering, and Sugih Jamin for their helpful comments on an earlier draft of this paper.

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] J. Davin and A. Heybey. *A Simulation Study of Fair Queueing and Policy Enforcement*, In **Computer Communication Review**, 20(5), pp 23-29, 1990.

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

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

[10] F. Dix, M. Kelly, and R. Klessig. *Access to a Public Switched Multi-Megabit Data Service Offering*, In **Computer Communication Review**, 20(3), pp 36-61, 1990.

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

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

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

[14] 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.

[15] S. Floyd. *Link-sharing, resource management, and the future internet*, preprint, 1993.

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

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

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

[19] 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.

[20] 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.

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

[22] 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.

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

[24] 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.

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

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

[27] 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.

[28] 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.

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

[30] 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.

[31] 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.

[32] 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.

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

[34] M. Steenstrup. *Fair Share for Resource Allocation*, 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. *A New Architecture for Packet Switching Network Protocols*, In **Technical Report LCS-TR-455**, Laboratory for Computer Science, Massachusetts Institute of Technology, 1989.

[39] L. Zhang. *VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks*, In **ACM Transactions on Computer Systems**, Vol. 9, No. 2, pp 101-124, May 1991. Also in Proc. ACM SIGCOMM '90, pp 19-29.