Link-Sharing and Resource Management Models for Packet Networks

Article *in* IEEE/ACM Transactions on Networking · September 1995

DOI: 10.1109/90.413212 · Source: IEEE Xplore

2 authors, including:

Sally Floyd
University of California, Berkeley
139 PUBLICATIONS **41,906** CITATIONS

SEE PROFILE

Link-sharing and Resource Management Models for Packet Networks

Sally Floyd

September 13, 1993

Abstract

This paper discusses the requirements and potentials for link-sharing mechanisms in packet networks and presents algorithms for hierarchical link-sharing. If multiple agencies, protocol families, or traffic types are to share the bandwidth on a link in a controlled fashion, the gateways need to provide some set of mechanisms for link-sharing. Link-sharing and realtime services are both emerging services that require resource management mechanisms at the gateway. Rather than requiring a gateway to implement separate mechanisms for link-sharing and for realtime services, the approach in this paper is to view link-sharing requirements and requirements for realtime services as simultaneous and in some respect complementary constraints at the gateway that can be implemented with a unified set of mechanisms.

The relationship between link-sharing requirements and real-time requirements at the gateway is discussed in this paper in some detail. It is not possible to completely predict the requirements of the real-time services that might evolve in the internet over the next decade. We argue that link-sharing is an essential component that can provide gateways with the flexibility to accomodate emerging real-time applications and new network protocols.

1 Introduction

There are a number of emerging requirements for resource management in the internet; these requirements include both link-sharing services and services for realtime traffic. In the absence of unlimited bandwidth, some realtime service at the gateway is desirable for traffic such as audio or video traffic that requires controlled delay. The need for link-sharing services has received less attention in the research community than the needs of realtime traffic. Link-sharing services are required when the bandwidth on a link is to be shared between different agencies, protocol families, or traffic types in a controlled fashion. This paper discusses the need for link-sharing, proposes algorithms for implementing link-sharing, and discusses the ways that appropriate link-sharing mechanisms can contribute to the implementation of flexible, efficient, and robust mechanisms for realtime services at the gateway.

One requirement for link-sharing is to share bandwidth on a link between multiple agencies, where each agency pays a fixed share of the costs, and each agency wants to receive a guaranteed share of the link bandwidth during congestion. At the same time, bandwidth that is not being used by one agency should be available to other agencies sharing the link. Another requirement for link-sharing might be to share bandwidth on a link between different protocol families, where controlled link-sharing is desired because the different protocol families have different responses to congestion. A third example for link-sharing is to share bandwidth on a link between different traffic types, such as telnet traffic, ftp traffic, or realtime traffic such as audio and video. All three of these examples of link-sharing explicitly deal with the aggregation of traffic on a link.

These various examples for link-sharing, taken together, naturally lead to a requirement for hierarchical link-sharing. For example, the bandwidth on a link might be shared between multiple agencies, and each agency might want to share its allocated bandwidth between several traffic types. This leads to a hierarchical *link-sharing structure* associated with an individual link in the network, with each *class* in the link-sharing structure corresponding to some aggregation of traffic (or in some cases to an individual connection). Different links are likely to have different link-sharing structures. Thus two connections that are aggregated into one class on one link might be in separate classes on the following link.

Link-sharing services and realtime services involve simultaneous sets of constraints to be satisfied at the gateway. Realtime services can involve throughput and delay commitments by the network that are negotiated in advance, and that require a fine-grained attention to packet scheduling. Link-sharing services are likely to involve somewhat less stringent guarantees that apply

over a somewhat longer time scale. This paper addresses the interaction between realtime services at the gateway and more general link-sharing services. Instead of implementing these two services with separate pieces of code, it is preferable to see these two services as integrated into a single set of mechanisms.

As an example, consider a link shared between two classes of realtime and nonrealtime traffic. In the absence of congestion at the link, the scheduler at the gateway could schedule the two classes of traffic using the most appropriate approach: possible examples include FIFO, strict priority, or round robin. However, if during congestion one of the two traffic classes began using more than its allocated share of the link bandwidth over some time interval, denying bandwidth to the other class of traffic, then the link-sharing mechanisms could be used to detect this impending problem. In this case the scheduler could call upon additional scheduling mechanisms to control the link bandwith used by the *over-limit* class. Thus, for example, the link-sharing mechanisms could work along with a priority-based scheduling mechanism for the realtime traffic.

Link-sharing can add flexibility to the resource management provisions of a network. For example, a network with link-sharing mechanisms at the gateways could implement a new transport protocol, with a less conservative response to congestion than that in TCP, and provide for controlled link-sharing between the two protocols. This controlled link-sharing would take care of concerns that the new transport protocol would "steal" all of the network bandwidth in times of congestion.

As a second example of the flexibility that link-sharing mechanisms can contribute to a gateway's resource management capabilities, consider the current need for some mechanism to protect data traffic in the Internet from the growing volume of realtime traffic on the M-Bone,[1] as well as the need to protect the realtime traffic from the delays caused by competing data traffic. The long-term solution to this problem involves a suite of realtime services that would include flow specifications for the realtime applications, a set-up procedure, and admissions control procedures, in addition to appropriate scheduling mechanisms at the gateway. However, while this long-term solution is not yet deployed, the requirement for controlling the realtime traffic could be met by implementing link-sharing mechanisms at the gateways, guaranteeing that over some relevant time interval both the realtime and the nonrealtime traffic can receive a share of the link bandwidth. These same link-sharing mechanisms could be used to provide for hierarchical link-sharing for the realtime traffic. For example, some fraction of the bandwidth "allocated" to the realtime traffic could itself be allocated to a particular subset of the realtime traffic, say to an audio- and video-conference of the IETF.

The approach to link-sharing in our paper is derived from the hierarchical class-based resource management proposals of Van Jacobson [CJ91]. The approach in [CJ91] proposes a set of flexible, efficiently-implemented gateway mechanisms that can meet a range of service and link-sharing requirements. Our paper is in the context of a continuing discussion about resource management in the Internet that involves contributions from a number of people [CJ91, CSZ92, SCZ93]. This paper was in part motivated as a response to the scheduling architecture proposed in [SCZ93]. We are in substantial agreement with many of the perspectives outlined in [SCZ93]; this paper elaborates on a particular disagreement over the relationship between link-sharing services and real-time services provided by the gateways. Thus, the framework for this paper borrows heavily from the framework in [SCZ93].

While we agree with the authors of [SCZ93] that a discussion of service models is necessary, that is not the specific task of this paper. This paper discusses the appropriate framework for implementing link-sharing services, and discusses the service model only where the service model is directly relevant to link-sharing issues.

Section 2 describes the link-sharing goals in more detail. Section 3 gives the general guidelines for implementing hierarchical link-sharing, given a resource management framework consisting of a general scheduler and a link-sharing scheduler. Section 4 explores some link-sharing guidelines that are approximations to the more rigorous guidelines outlined in Section 3. Section 5 shows some simulations of link-sharing at the gateway. Section 6 discusses the relationship between the link-sharing goals and the goals for realtime traffic. Section 7 compares the link-sharing framework discussed in this paper with other proposals for link-sharing in the literature. Section 8 gives conclusions and discusses related work. [In general, I have not finished adding references to this paper, acknowledging other work.]

2 The link-sharing goals

This section discusses link-sharing goals in more detail. The link-sharing goals are to share the bandwidth on a single link between multiple agencies, protocol families, or traffic types. In cases where several organizational agencies share the cost of a link, they might want

[1] The M-Bone is the current multicast backbone overlaid over parts of the Internet. The first large-scale audioconference is described in [CD92].

to share the bandwidth on the link in a controlled but flexible fashion. Similarly, for a link shared by different protocol families with different responses to congestion, controlled link-sharing can be desirable to ensure that the protocol family with the more "aggressive" response to congestion does not capture all of the link bandwidth in times of congestion. For a link shared by realtime and nonrealtime traffic, controlled link-sharing can ensure that one class of traffic is not allowed to starve out the other class over some appropriate time interval.

Thus while there are a number of different motivations for link-sharing in the network, the requirements for link-sharing are essentially the same, whether the link-sharing is between service classes, organizations, or protocol families. Instead of having separate pieces of code in the gateway for each of these link-sharing situations (and again for the provision of realtime service), we would argue that a single set of mechanisms for link-sharing should be implemented and carefully coordinated with any additional mechanisms for providing realtime service.

First consider link-sharing for a link with a single flat link-sharing structure, as in Figures 1 and 2. For the link-sharing structure in Figure 1, the link is shared by a number of realtime and non-realtime traffic *classes*. The audio and video classes are examples of *leaf* classes in the link-sharing structure, and the aggregated link class is an *interior* class. This link-sharing structure specifies the desired division of bandwidth in times of congestion for a particular point-to-point link in a network. In Figure 1 the *telnet* class could be a class of delay-sensitive traffic such as X and NFS traffic as well as telnet traffic. Similarly, the *mail* class could be a class of delay-insensitive traffic such as FAX as well as mail traffic.

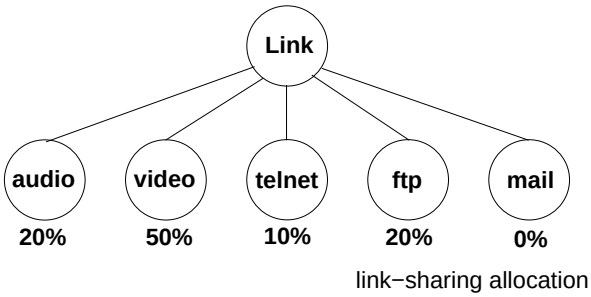

Figure 1: Link-sharing between service classes.

For a flat link-sharing structure such as the one in Figure 1 the link-sharing requirements are fairly straightforward. A *link-sharing bandwidth* is allocated to each class (expressed in Figure 1 as a fraction of the overall link bandwidth). The first link-sharing goal is that each class with sufficient demand should be able to receive roughly its allocated bandwidth, over some interval of time, in times of congestion. As a consequence of this link-sharing goal, in times of congestion some classes might be *restricted to* their link-sharing bandwidth. For a class with a link-sharing allocation of zero, such as the 'mail' class in Figure 1, the bandwidth received by this class is determined by the other scheduling mechanisms at the gateway; the link-sharing mechanisms do not "guarantee" any bandwidth to this class in times of congestion.

The link-sharing goals are a rough quantitative bandwidth commitment by the network. Associated with these link-sharing goals is some notion of the time interval over which the link-sharing goals apply. For example, for the link-sharing between service classes in Figure 1, it might be considered unacceptable if all telnet and ftp traffic was denied service for minutes at a time. Nevertheless, over very small time intervals the packets can be scheduled by other scheduling mechanisms that take into account the different service goals of the various classes. There is no goal that each service class should receive its link-sharing bandwidth over arbitrarily-small time intervals.

Definitions: exempt classes. It is possible that some classes could be marked as *exempt* from link-sharing enforcement; for these classes, the specified link-sharing bandwidth would never be used by the scheduler to restrict traffic from that class. If this is the case, then either the general scheduler and the admissions control procedure should ensure that the traffic from these classes does not violate the link-sharing goals, or there should be a clear understanding that the scheduling of this traffic takes precedence over the link-sharing goals.[2]

Definitions: restricted classes. It is also possible that some classes could be marked as *restricted*, and would be restricted by the link-sharing mechanisms to their link-sharing bandwidth regardless of the congestion or lack of congestion at the link. This could be the case, for example, for a traffic class where reliable delay is more important than low delay.

The second link-sharing goal is that when some class is not using its allocated bandwidth, then the distribution of the 'excess' bandwidth among the other classes should not be arbitrary, but should follow some appropriate set of guidelines. For a flat link-sharing structure, this distribution of excess bandwidth is completely determined by the other scheduling mechanisms used at the gateway, and is not specified by the link-sharing structure. For example, for link-sharing between service classes as in Figure 1, the distribution of excess bandwidth might depend on the relative priority and

[2]The proposal in [SCZ93, JSZC92] is for the admissions control procedure to be used to ensure that the real-time traffic does not violate the link-sharing goals.

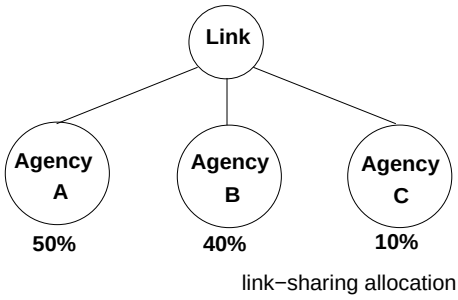

Figure 2: Link-sharing between multiple agencies or protocol families.

service needs of the classes. However, consider link-sharing between agencies or between protocol families, as in Figure 2. If agency A has little traffic to send, agency B might consider it unfair or arbitrary if all of the 'excess' bandwidth was given to agency C. For link-sharing between agencies or between protocol families, the ideal would be for the scheduling mechanisms to distribute any 'excess' bandwidth in a way that takes into account the relative link-sharing allocations of those entities.

Multiple link-sharing constraints at a gateway can be expressed by a hierarchical link-sharing structure such as in Figure 3. This link-sharing structure specifies link-sharing between agencies, between protocol families, and between service classes. All arriving packets at the gateway are assigned to one of the *leaf* classes; the *interior* classes are used to designate guidelines about how 'excess' bandwidth should be allocated. Thus, the goal is that the three service classes for agency A should collectively receive 50% of the link bandwidth over appropriate time intervals, given sufficient demand. If the realtime class for agency A has little data to send, the hierarchical link-sharing structure specifies that the 'excess' bandwidth should be allocated to other subclasses of agency A.

The link-sharing goals can be summarized as follows:
Link-sharing goals:
1: Each interior or leaf class should receive roughly its allocated link-sharing bandwidth over the specified time intervals, given sufficient demand.
2: If all leaf and interior classes with sufficient demand have received at least their allocated link-sharing bandwidth, the distribution of any 'excess' bandwidth should not be arbitrary, but should follow some set of reasonable guidelines. □

Definitions: general scheduler, link-sharing scheduler, regulated classes. The implementation of the first link-sharing goal is discussed in detail in this paper. The framework in this paper assumes that the scheduling mechanisms at the gateway include a *general scheduler*, which might schedule packets from classes without regard to the link-sharing structure, and a *link-sharing scheduler* that schedules packets from some classes that have been exceeding their link-sharing allocations in times of congestion. One job of the general scheduler is to provide for realtime traffic that has particular delay or throughput requirements. We call a class a *regulated* class when packets from that class are scheduled by the link-sharing scheduler at the gateway; and we call the class *unregulated* when traffic from the class is scheduled by the general scheduler. Guidelines in Section 3 specify when a class can continue unregulated and when the class should become regulated by the link-sharing scheduler.

The second part of the link-sharing goal concerning the distribution of excess bandwidth is not addressed in this paper, but has to be taken into account by the general scheduler. In our simulator, the general scheduler distributes 'excess' bandwidth to higher priority classes, with the distribution proportional to the relative link-sharing allocations of those classes.

Note that these link-sharing goals are quite modest. The link-sharing mechanisms in this paper constitute the minimum action possible to allow classes to receive their allocated link-sharing bandwidth over the relevant time intervals. The link-sharing mechanisms do not attempt to rate-allocate classes in the absence of congestion, to reshape traffic, or to specify precisely the bandwidth to be received by each class given the current demand.

As an example of the minimum of action implied by the link-sharing goals, the link-sharing structure at a gateway could be to allocate 80% of the link bandwidth to a realtime class of traffic, and 20% to the non-realtime class. This could be coupled with a priority-based general scheduler that gives priority to the realtime traffic, and an admissions control procedure for realtime traffic with the goal of limiting the realtime traffic to 50% of the link bandwidth. In this case, the link-sharing scheduler would only be used to regulate the bandwidth of the realtime class if the realtime class in fact exceeded 80% of the link bandwidth over some interval of time while the non-realtime class had unsatisfied demand. This is unlikely to happen, given the admissions control procedure, so in fact the link-sharing scheduler might never be needed. But the presence of the link-sharing mechanisms ensures that the non-realtime class will not be denied its allocated bandwidth for long intervals of time.

The link-sharing structure for a particular link can have both static and dynamic components. A static link-sharing structure would be appropriate when a network administrator designates that a particular link should be shared between multiple agencies; in this case, the link-

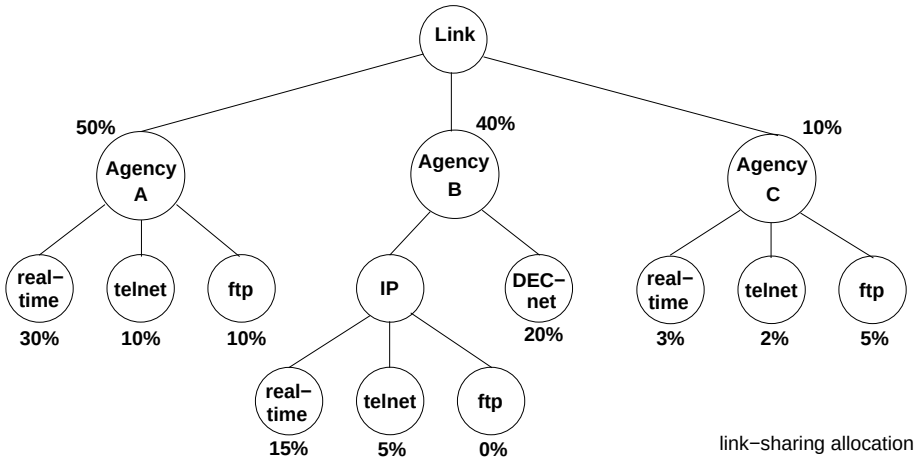

Figure 3: A hierarchical link-sharing structure.

sharing bandwidth allocated to each agency might be set by the network administrator. A link-sharing structure with dynamic components would be appropriate when the link-sharing mechanism is used to ensure that particular realtime connections receive their required share of the link bandwidth (at the same time ensuring that the realtime connections don't monopolize the bandwidth on a link). For dynamic link-sharing, mechanisms are needed to limit the lifetimes of the dynamic components. The issue of lifetime restrictions is not addressed in this paper, but needs to be considered in the context of set-up protocols and admissions control procedures for realtime traffic.

Note that the link-sharing goals don't attempt to completely address the questions of congestion control at the gateway. The link-sharing mechanisms monitor and control bandwidth allocations between various classes of traffic; the question of congestion control for the traffic within the class remains. For a leaf class that consists of a single connection, there is a question of how much link-sharing bandwidth and buffer capacity needs to be allocated to that class. This depends on the general scheduler at the gateway. For leaf classes that contain a number of aggregated connections, congestion control could be provided by an end-to-end transport protocol such as TCP, or by an explicit admissions control procedure for that class, or perhaps by some form of rate-adaptive congestion control for some classes of realtime traffic. The framework in this paper assumes that each class has its own queue. For some classes, the gateway could monitor the average queue size and/or the traffic patterns for the class, and provide appropriate feedback to the sources [FJ93].

The ability to explicitly control traffic aggregation at the gateway is one of the strengths of link-sharing mechanisms. The link-sharing goals require a data structure associated with each link, describing the class structure at that link, and giving the link-sharing bandwidth for each class.

One required link-sharing mechanism is a *classifier* at the gateway to classify arriving packets to the appropriate class. Another required link-sharing mechanism in our framework is an *estimator* to estimate the bandwidth used by each class over the appropriate time interval, to determine whether or not the class has been receiving its link-sharing bandwidth.

3 Formal link-sharing guidelines

This section gives formal guidelines for implementing hierarchical link-sharing at the gateway. Section 4 discusses implementations might approximate these formal link-sharing guidelines.

The framework of this paper assumes that the gateway has both a *general scheduler*, a general scheduling mechanism that is used in particular to accommodate the needs of real-time traffic, and a *link-sharing scheduler* to regulate classes that are violating the link-sharing goals in times of congestion. This paper does not specify the general scheduler in more detail: the general scheduler could use strict priority between different classes of traffic, with some mechanism for scheduling traffic from different classes with the same priority, or the general scheduler could use more complex scheduling methods, such as 'just-in-time' scheduling for some real time traffic [SCZ93].

In the absence of persistent congestion (or the absence of persistent congestion for all but the lowest-priority classes), the general scheduler should be all that is required to schedule traffic on the output link. However, in the presence of congestion the scheduler might also

have to take into account link-sharing goals for sharing the link bandwidth among different traffic types, protocol families, or agencies.

This paper proposes a framework for satisfying the simultaneous constraints of the realtime service goals and the link-sharing goals. The link-sharing guidelines in this section specify when a class can continue unregulated, scheduled by the general scheduler, and when the class should be regulated by the link-sharing scheduler.

This paper focuses on the interaction between the general scheduler and the link-sharing scheduler. We do not specify the general scheduler or the link-sharing scheduler in more detail; the general scheduler could be one of a number of proposed scheduling algorithms that determines the packet-by-packet scheduling necessary to meet the service goals, and the link-sharing scheduler could use one of a number of algorithms to restrict the bandwidth of regulated classes. The implementations in our simulator are discussed in the appendix.

The link-sharing scheduler could take one from a range of actions to regulate a class. One option for the link-sharing scheduler is to rate-limit the regulated class to its link-sharing bandwidth, or to some larger bandwidth that takes into account currently-inactive classes; this regulation would be accompanied by some strategy for dropping arriving packets when necessary. However, there are other options; for example, the link-sharing scheduler could decrease the priority of the regulated class, so that the general scheduler schedules packets from that class less frequently. As another option, the link-sharing scheduler could dynamically change the link-sharing allocations of the various classes.

The link-sharing scheduler does not have to be a separate section of code from the general scheduler. For example, as the appendix explains, in our simulator the "link-sharing scheduler" simply uses a different rule that the "general scheduler" is setting a class parameter called the *time-to-send* field, indicating the next time that a packet is allowed to be sent from that class.

Definitions: overlimit, underlimit, at-limit. A class is called *overlimit* if it has recently received more than its allocated link-sharing bandwidth over a particular time interval, *underlimit* if it has received less that a specified fraction of its link-sharing bandwidth over that time interval, and *at-limit* otherwise. □

The limit status of each class is determined by an *estimator*, given the relevant time interval. Note that the specification of the time interval or time constant is critical to the definition of the limit status; this time interval determines the interval over which the gateway attempts to enforce the link-sharing guidelines. Note that the root node of the link-sharing structure, representing the link itself, by definition can be at-limit or underlimit but not overlimit.

One possible method for implementing the estimator is discussed in the appendix. Some methods of implementing the estimator can allow a previously-idle class to send a large burst of traffic before that class is estimated as *overlimit*. The procedure for implementing the estimator can be modified to prevent this, explicitly limiting the 'credit' that a class receives for having been idle for a long period, or limiting the 'penalty' that a class receives for having used 'excess' bandwidth. (If the link-sharing scheduler ensures that overlimit classes still receive their allocated share of the link bandwidth, this also reduces the 'penalty' that a class receives for having used 'excess' bandwidth in the past.)

The limit status of the classes is used to determine whether or not explicit action has to be taken to correct the link-sharing behavior of the traffic.

Definitions: satisfied, unsatisfied. A leaf class is defined as *unsatisfied* with the link-sharing behavior if it is underlimit and has a persistent backlog, and *satisfied* otherwise. A non-leaf class is defined as *unsatisfied* with the link-sharing behavior if it is underlimit and has some descendant class with a persistent backlog. □

We do not define a *persistent backlog* in more detail; the intention is that an unsatisfied class is a class that is underlimit and that has sufficient demand to use additional bandwidth. An underlimit class that occasionally has a packet or two in the queue for a brief time is not in fact unsatisfied with the link-sharing behavior. In our simulator's implementation of the formal link-sharing guidelines in this section, any class with a non-empty queue is defined as having a persistent backlog. (The notion of a persistent backlog is not used at all in the approximations to the formal link-sharing guidelines presented in the next section.)

In proposing formal link-sharing guidelines, we first consider a link with a flat link-sharing structure, as in Figure 1. In this case, the link-sharing guidelines are fairly clear and intuitive. When a class is not overlimit *or* when there are no unsatisfied classes, then the class does not need to be regulated by the link-sharing scheduler. However, when a class is overlimit *and* some other class is unsatisfied, then the overlimit class is contributing to congestion on the link, and should be regulated by the link-sharing scheduler. This regulation should continue until the class is no longer overlimit or until there are no more unsatisfied classes.

Given these guidelines for link-sharing, an overlimit class is only regulated when some other class has unfilled demand and has not been receiving its allocated bandwidth over the specified time interval. When all classes are satisfied with the link-sharing behavior, no classes have to be regulated at the gateway. For a hierarchical link-sharing structure, the link-sharing guidelines

are an extension of the guidelines given above. The Formal link-sharing guidelines given below implement the link-sharing goals. The time interval over which the link-sharing goals are applied is determined by the time constants used in calculating the limit status of each class.

Formal link-sharing guidelines:
A class can continue unregulated if one of the following conditions hold:
1: The class is not overlimit, OR
2: The class has a not-overlimit ancestor with no unsatisfied descendants.
Otherwise, the class will be regulated by the link-sharing scheduler. □

Note that these link-sharing guidelines are used simply to decide if a class is allowed to be scheduled by the general scheduler, *unreguled*, or whether the class should have its bandwidth *regulated* by the link-sharing scheduler. The link-sharing guidelines to not explicitly guarantee particular fractions of the link bandwidth to unregulated classes. The actual division of the available bandwidth between the unregulated classes is determined by the general scheduler. The link-sharing guidelines are simply used to determine when some class is using more than its allocated bandwidth, and thereby contributing to the unsatisfied state of some other class in the link-sharing. A detailed analysis of the bandwidth distribution has to depend on the details of the general scheduler and link-sharing scheduler.

The cases below, illustrated in Figure 4, illustrate the Formal link-sharing guidelines.

Case 1: Consider a link-sharing hierarchy where no classes are overlimit. In this case no classes need to be regulated.

Case 2: In this case the Agency A realtime class is overlimit, but no classes are unsatisfied. From the link-sharing guidelines, there is no need for any classes to be regulated.

Case 3: In this case there are two overlimit classes, but only the Agency B non-realtime class is unsatisfied, and the Agency A class is not overlimit. From the link-sharing guidelines, only the Agency B realtime class needs to be regulated. Because Formal link-sharing takes into account the 'satisfied' or 'unsatisfied' status of related classes, the Formal link-sharing guidelines can distinguish between the two realtime classes in this case.

Case 4: In this case the Agency A class and the Agency A realtime class are overlimit, while the Agency B class and Agency B non-realtime class are unsatisfied. From the link-sharing guidelines, the Agency A realtime class is regulated. Because the Formal link-sharing guidelines take into account the limit status of ancestor classes, hierarchical link-sharing can be provided.

Case 5: In this case no leaf classes are unsatisfied, but the Agency A class itself is unsatisfied. Because the Agency A class is underlimit and has no unsatisfied descendants, the Agency A non-realtime class can continue unregulated. However, according to the link-sharing guidelines the Agency B realtime class should be regulated. This is another example of how the link-sharing guidelines can provide hierarchical link-sharing.

Case 6: This case is a slight variant of Case 5. The Agency A class is unsatisfied, and according to the link-sharing guidelines the Agency B realtime class should be regulated.

Case 7: This case differs from Case 6 in that the Agency A ftp class has accumulated a persistent backlog. From the link-sharing guidelines, all three overlimit leaf classes should be regulated.

Note that we did not specify in the link-sharing guidelines how frequently the scheduler should check whether or not a class needs to regulated. In our implementation the general scheduler checks whether or not a class can continue to send unregulated just before transmitting a packet from that class, but this check could also be made less frequently.

When should the control of a class by the link-sharing scheduler be terminated? One possibility is that a regulated class should remain regulated as long as the Formal link-sharing guidelines are not met. A class might oscillate frequently between being regulated and being unregulated, but this is not necessarily a problem. If for implementation reasons it is desired to reduce the frequency of these oscillations, the following guidelines could be used:

Alternate link-sharing guidelines:
A class can continue unregulated if one of the following conditions hold:
1: The class is not overlimit, OR
2: The class has a not-overlimit ancestor with no unsatisfied descendants.
Otherwise, the class will be regulated by the link-sharing scheduler.
A regulated class will continue to be regulated until one of the following conditions hold:
1: The class is underlimit, OR
2: The class has an underlimit ancestor with no unsatisfied descendants.
□

To summarize the link-sharing guidelines, when all classes are satisfied then no classes have to be regulated. Similarly, any class that is not overlimit can continue unregulated. However, when there is an unsatisfied class, some overlimit class will be regulated by the link-sharing scheduler.

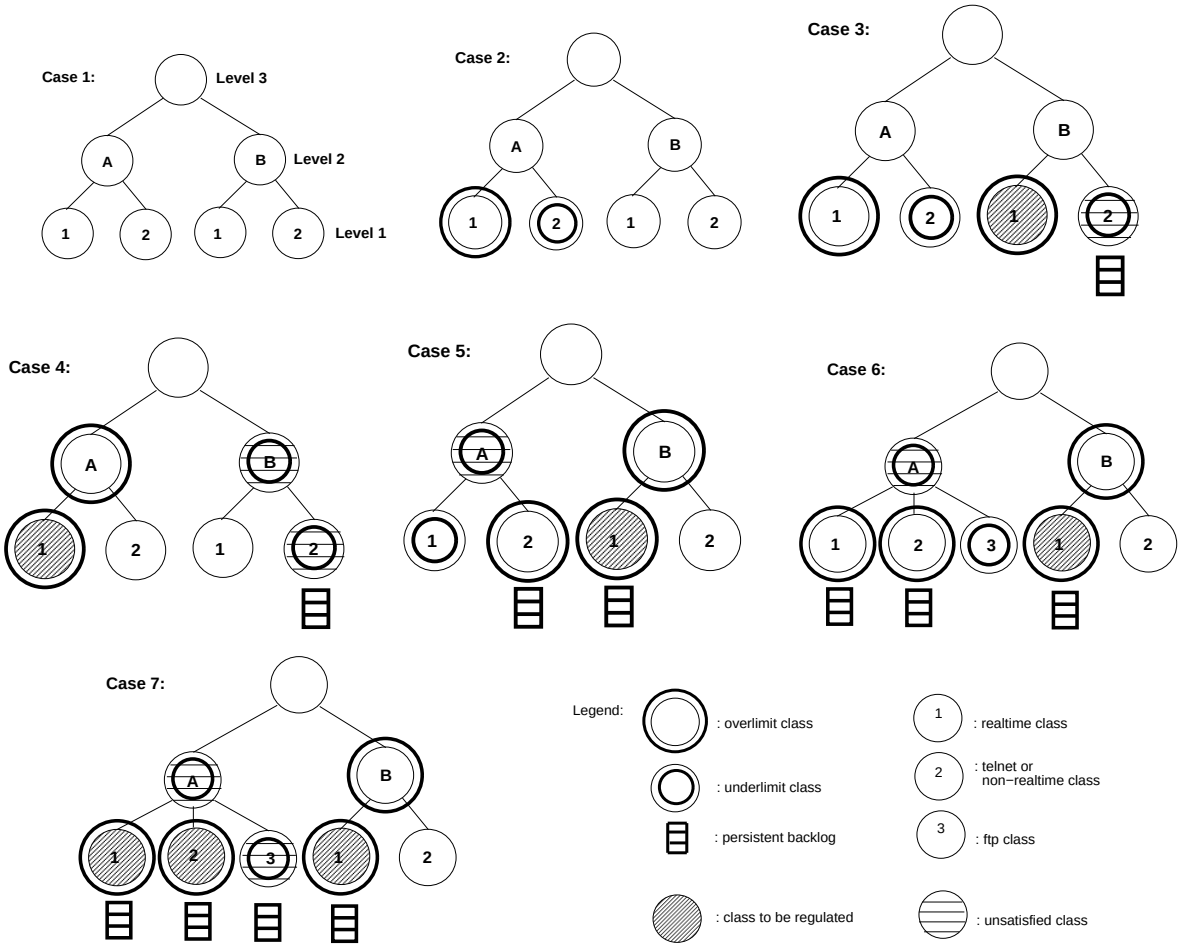

Figure 4: Examples of link-sharing scenarios.

4 Approximations to the Formal link-sharing guidelines.

The previous section describe a set of Formal link-sharing guidelines. With the formal guidelines the decision whether or not to regulate a class requires examining not only the limit status of parent classes but also the 'satisfied' status of descendants of those parent classes. This section explores several approximations to Formal link-sharing that can perhaps be more efficiently implemented. First we define Ancestors-Only link-sharing guidelines, where the decision whether or not to regulate a class requires examining only the limit status of parent classes. We discuss the capabilities and limitations of this approach. Next we define Top-Level link-sharing guidelines, which give somewhat improved performance over the Ancestors-Only link-sharing guidelines. The following section gives simulations illustrating Formal, Ancestors-Only, and Top-Level link-sharing.

The Ancestors-Only link-sharing guidelines are as follows:

Ancestors-Only link-sharing guidelines:
A class can continue unregulated if one of the following conditions hold:
1: The class is not overlimit, OR
2: The class has an underlimit ancestor.
Otherwise, the class will be regulated by the link-sharing scheduler. □

One attraction of this implementation would be that it is easy to check if a class needs to be regulated, simply by checking the limit status of the class and of the class's ancestors. Case 3 in Figure 4 illustrates why an overlimit class with at-limit ancestors should be regulated, given an Ancestors-Only approach. One drawback of the Ancestors-Only approach is that because the 'satisfied' status of sibling classes is not examined, no distinction can be made between the two realtime classes in Case 3. Either some class will be regulated unnecessarily, or the link-sharing goals will not be satisfied.

While Ancestors-Only link-sharing gives acceptable

results in most occasions, it is not as robust as Formal link-sharing. Ancestors-Only link-sharing requires some mechanism to prevent higher-priority classes from remaining unregulated in the presence of unsatisfied classes. The mechanism in Ancestors-Only link-sharing is to allow an overlimit class to remain unregulated only when some ancestor class is *underlimit* (instead of simply being *not overlimit*). Clearly for a flat link-sharing structure, given an Ancestors-Only approach, an overlimit leaf class cannot remain unregulated when the root class is at-limit and the output link is at full capacity; if this were the case, then an overlimit class could never be regulated. Therefore given an Ancestors-Only approach, the estimator has to distinguish between an at-limit and an underlimit ancestor class, allowing an overlimit leaf class to remain unregulated only when it has an underlimit ancestor. As a result, Ancestors-Only link-sharing is sensitive to the quantitative parameter used to distinguish between an at-limit and an underlimit class.

Ancestors-Only link-sharing can be sensitive to other parameters as well. One mechanism to limit the time that a class can remain unregulated in the presence of unsatisfied classes is to have the estimator limit the 'credit' that a class receives for having been idle for a period of time. This limits the number of back-to-back packets that could be sent from a previously-idle class before the class is classified as no longer underlimit. With Ancestor-Only link-sharing, it is similarly helpful to limit the 'credit' that an interior class receives for having been idle. This helps to limit the time that an overlimit child class can continue unregulated because of an underlimit parent class (which might also have a lower-priority unsatisfied child). Because Ancestors-Only link-sharing has no mechanism to block one class from 'borrowing' from an underlimit parent when some other class could send without borrowing, the performance of Ancestors-Only link-sharing is sensitive to the exact settings of the parameters for computing the overlimit and the underlimit status for interior classes.

Consider the link-sharing structure in Case 5 in Figure 4, and assume that the general scheduler gives priority to realtime traffic over non-realtime traffic. Assume further that the Agency A realtime class has little data to send for an extended period of time, and that the Agency B realtime class has unfilled demand. With Ancestors-Only link-sharing, the Agency B realtime class is allowed to send unregulated whenever the root class is underlimit, regardless of the limit status of Agency A. The result is that both Agency A and the root class will cycle between being underlimit and being not-underlimit. As a consequence Agency A's pattern of traffic on the output link is likely to be somewhat bursty. If Agency A is limited in the 'credit' that it gets for being idle for periods of time, then Agency A could receive less than its allocated link-sharing bandwidth.

The Top-Level link-sharing guidelines are a slight modification of the Ancestors-Only approach and give a more robust approximation to Formal link-sharing. In the Top-Level guidelines, the gateway still doesn't check the 'satisfied' status of sibling classes, as is done in Formal link-sharing. Instead, with the Top-Level approach each *level* in the hierarchical link-sharing structure is labeled, with the leaf classes labeled 'Level 1', the parent classes labeled 'Level 2', and so on, as is shown in Case 1 in Figure 4. The gateway maintains a *Top-Level* variable that indicates the highest level from which a class is allowed to 'borrow' bandwidth, and various heuristics are used to set and reset the *Top-Level* variable.

Top-Level link-sharing guidelines:
A class can continue unregulated if one of the following conditions hold:
1: The class is not overlimit, OR
2: The class has an underlimit ancestor whose level is at most *Top-Level*.
Otherwise, the class will be regulated by the link-sharing scheduler. □

The following list gives the set of heuristics used in our simulator for setting the Top-Level variable.
Heuristics for setting the Top-Level variable:
1: If a packet arrives for a not-overlimit class, set *Top-Level* to 1.
2: If *Top-Level* is i, and a packet arrives for an overlimit class with an underlimit parent at a lower level than i (say j), then set *Top-Level* to j.
3: After a packet is sent from a class, and that class now either has an empty queue, or is unable to continue unregulated, then set *Top-Level* to *Infinity*.

A number of heuristics could be used to set and reset the *Top-Level* variable. The gateway should set *Top-Level* to 1 only when the gateway knows that some not-overlimit class has a packet to send; the gateway should set *Top-Level* to i only when the gateway knows that some class can send a packet because of an underlimit ancestor at level at most i. Because heuristics are used, the setting of the *Top-Level* variable can reflect some partial knowledge of the gateway. The *Top-Level* mechanism avoids the overhead of Formal link-sharing of checking the 'satisfied' status of all descendants of underlimit classes, while still limiting the time that an overlimit class can remain unregulated simply because of an underlimit ancestor.

While Top-Level link-sharing requires additional overhead in maintaining the *Top-Level* variable, there are other ways in which Top-Level link-sharing requires less overhead than Ancestors-Only link-sharing. For

example, in Top-Level link-sharing when the *Top-Level* variable is one, then the scheduler doesn't check the limit status of parent classes before deciding whether or not a class needs to be regulated.

5 Link-sharing simulations

This section describes simulations that illustrate the performance of Formal, Ancestors-Only, and Top-Level link-sharing. The simulation scenarios are quite simple, with only one congested gateway in each scenario.

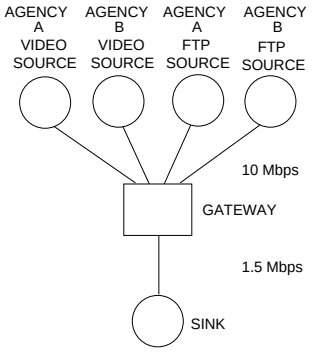

Figure 5: Simulation scenario for two-agency link-sharing.

Figure 5 shows the network scenario for one of the simulations. Each simulation network has a single congested gateway, with the link-sharing structures for the congested link shown in Figures 6 through 8. Each class has a single constant-bit-rate source with its own input link to the congested gateway, and each class has sufficient demand to use the entire link bandwidth of the shared link. These (admittedly unrealistic) sources are used as a simple way to explore link-sharing free from extraneous influences. In these simple simulations the traffic for each class is generated by a single source, but in general the traffic in a class could consist of many connections from many different sources. Each leaf class has its own fixed-size queue at the gateway, and arriving packets are dropped when the queue is full. In these simulations, somewhat arbitrarily, the ftp packets are 1000 bytes, the video packets are 190 bytes, and the audio packets range from 250 to 500 bytes.

Each leaf class in the link-sharing structure is assigned a priority level as well as a link-sharing allocation. The general scheduler in our simulator uses strict priority. For classes of the same priority the general scheduler uses a variant of weighted round-robin, with weights proportional to the link-sharing bandwidths of the classes. Thus within a priority level the general scheduler distributes bandwidth according to the link-sharing allocations of the classes. The link-sharing

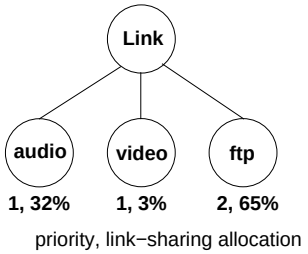

Figure 6: Link-sharing structure for the simulation of flat link-sharing.

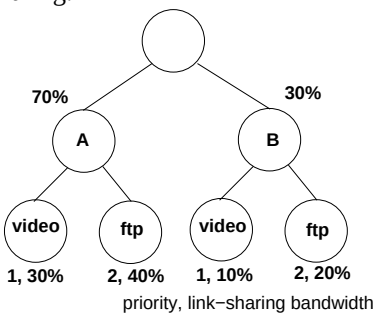

Figure 7: Link-sharing structure for the simulation of two-agency link-sharing.

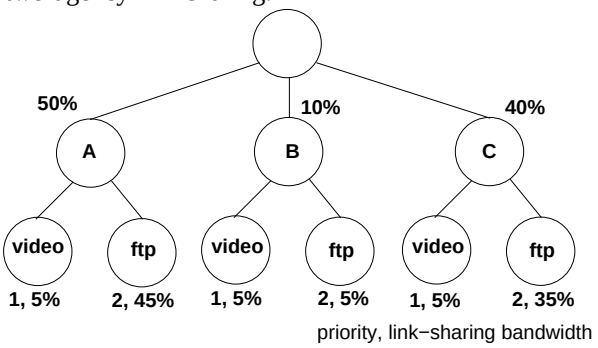

Figure 8: Link-sharing structure for the simulation of three-agency link-sharing.

scheduler in our simulator rate-limits each regulated class to its link-sharing bandwidth. The estimator, general scheduler, and link-sharing scheduler in the simulator are explained in more detail in the appendix. The time constant used by the simulator's estimator for computing the limit status for a class is relatively small, equal to the time to transmit 16 packets from the class.[3]

For the simulation in Figure 9, the class structure for the congested link has two high-priority classes and one lower-priority class. In Figure 9 the x-axis shows time, and a line gives the percentage of the link bandwidth used by each class. The link bandwidth is calculated (for the graph) over one-second intervals. In the sim-

[3]The *time constant* for the estimator is defined in Appendix A.

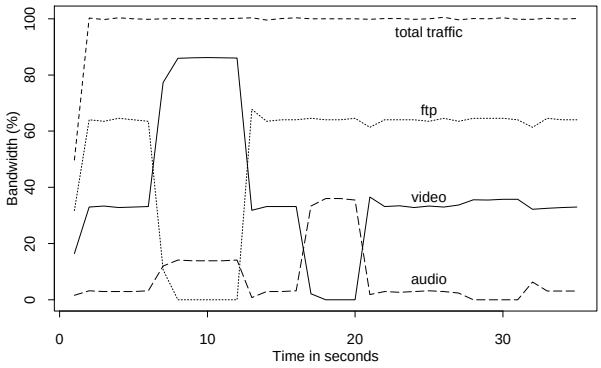

Figure 9: Flat link-sharing with Top-Level link-sharing guidelines.

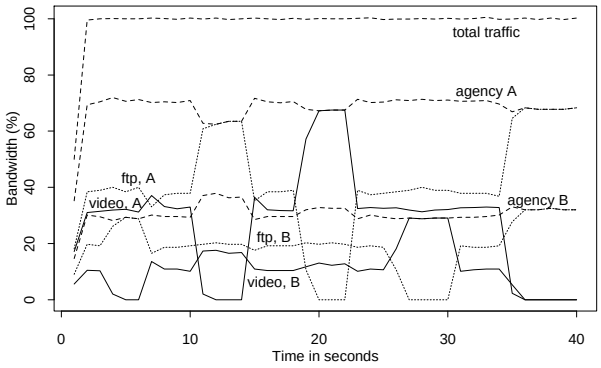

Figure 11: Two-agency link-sharing with Ancestor-Only link-sharing guidelines.

ulation, for each traffic class in turn the source stops transmitting for some seconds. The simulation shows that when all sources are transmitting, each class receives roughly its link-sharing allocation. When the ftp class stops transmitting for a few seconds, the 'excess' bandwidth is shared between the two realtime classes in proportion to the link-sharing bandwidths of those two classes. (This is because of the weighted round-robin used by the general scheduler.) When the video class stops transmitting for a few seconds, the 'excess' bandwidth is used by the audio class, because the general scheduler gives the audio class priority over the ftp class. Similarly, when the audio class stops transmitting for a few seconds, the small measure of 'excess' bandwidth is used by the video class. Even though the general scheduler uses priority scheduling, the link-sharing mechanisms ensure that the ftp class receives at least its link-sharing bandwidth when it has sufficient demand. This simulation gives essentially the same results with Formal, Ancestor-Only, or Top-Level link-sharing.

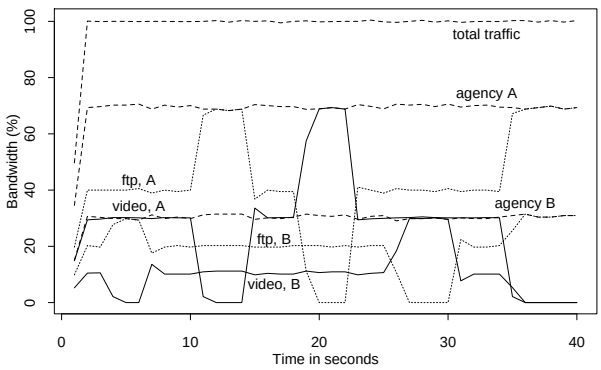

Figure 10: Two-agency link-sharing with Formal link-sharing guidelines.

The second set of simulations shows two agencies

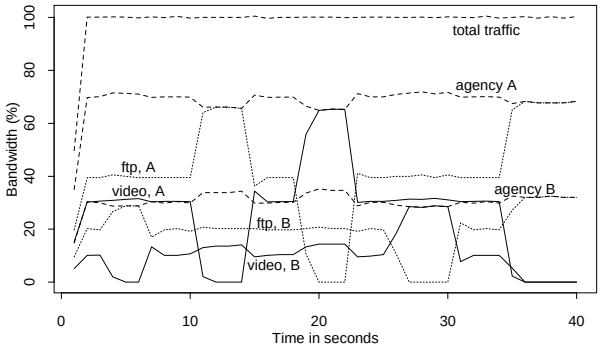

Figure 12: Two-agency link-sharing with Top-Level link-sharing guidelines.

sharing a congested link, with two traffic classes for each agency. The simulations in Figures 10 through 12 show simulations with Formal, Ancestor-Only, and Top-Level link-sharing. The bandwidth used by the two video classes is represented by solid lines, the bandwidth used by the two ftp classes is represented by dotted lines, and the total bandwidth used by each interior class is represented by a dashed line. In each simulation the source for each class stops transmitting for some seconds. Note that when some class stops transmitting, the 'excess' bandwidth is used by the other class in the same agency, as the hierarchical link-sharing structure specifies. Thus, each agency receives roughly its link-sharing bandwidth as long as it has sufficient demand. At the end of the simulation, the video sources stop transmitting again, and each ftp class receives the link-sharing allocation of its parent class.

Note that in the simulation with Formal link-sharing, each agency receives its link-sharing allocation throughout the simulation. In the simulation with Ancestor-Only link-sharing agency A receives less than its link-sharing allocation for part of the simulation.

This problem is partly corrected in the simulation with Top-Level link-sharing.

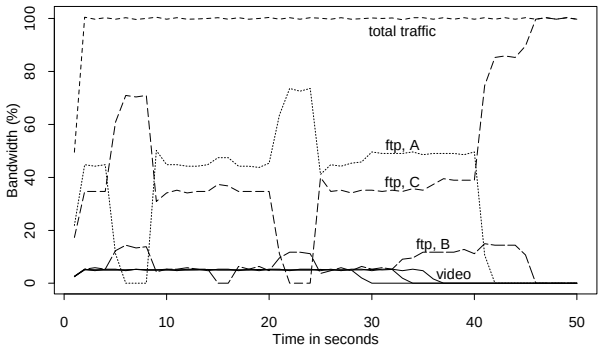

Figure 13: Three-agency link-sharing with Top-Level link-sharing guidelines.

The third simulation shows three agencies sharing a congested link. In this simulation all three video classes are marked as *restricted* and are not allowed to use bandwidth from parent classes. Thus each video class receives at most 5% of the link bandwidth regardless of other traffic on the link. This simulation shows that when the source for one of the ftp classes stops transmitting for a few seconds, the 'excess' bandwidth is distributed between the other two ftp classes, roughly in proportion to the link-sharing allocations of those classes. (Again, this is because of the weighted round-robin used by the general scheduler.) At the end of the simulation, the sources for several of the classes stop altogether. These simulation results are essentially the same for Formal, Ancestor-Only, and Top-Level link-sharing.

In general, in simulations with Ancestor-Only link-sharing the higher-priority classes sometimes receive slightly more bandwidth than is allocated, and the agency-level link-sharing is sometimes imprecise. The Ancestor-Only link sharing is also more sensitive to the setting of the various parameters used in computing the limit status of the interior classes. The problems are reduced with Top-Level link-sharing, and these problems do not occur in the simulations with Formal link-sharing.

[I am going to add some simulations to illustrate what might occur if link-sharing was used along with a priority-based scheduler (with priority for realtime traffic) to control M-Bone traffic in the Internet. The simulation will show the low delay that the realtime traffic gets as long as the realtime bandwidth stays within the link-sharing guidelines. The simulation will show that when the realtime traffic exceeds the link-sharing guidelines, then the realtime traffic is limited by the link-sharing scheduler to its link-sharing bandwidth. Because the buffer space allocated to the realtime traffic will be somewhat modest, when the realtime traffic is limited to its link-sharing bandwidth the delay will remain relatively low, but the packet drop rate for the realtime traffic will increase as the arrival rate of the realtime traffic increases. The realtime class will be divided into two classes, an IETF class and a non-IETF class, perhaps of the same priority but each with its own link-sharing bandwidth. Thus, this simulation will highlight the ways that link-sharing can be integrated with a priority-based scheduling mechanism.]

6 Link-sharing and realtime traffic

This section discusses briefly the question of service models for realtime traffic, paying particular attention to the relationship between link-sharing goals and realtime service models. We emphasize the need for some flexibility in defining future service models; it will be hard to predict completely the new service models that will be required by emerging new applications in the Internet. As an example, one possibility for an emerging new application is variable-bit-rate video that requires a certain minimum bandwidth, but that adapts its source coding rate to fit current network conditions. A related possibility is video with drop-preference, where the video coding is designed to accommodate a certain level of packet drops in the network. Link-sharing mechanisms at the gateway can be a key component in allowing flexibility in accommodating new service models in the internet.

Discussions of service models have included proposals for priority for realtime over non-realtime traffic. Several researchers have proposed methods for delivering guaranteed throughput and low delay for realtime traffic, with an appropriate admissions control procedure. Many of these service models assume that realtime connections would negotiate for a particular class of service (including, for example, a certain average or maximum delay, or allowing for a certain statistical fraction of packet drops).

A new service model was introduced in [CSZ92] of *predictive service* for loss-tolerant applications with adaptive playback times. This model was motivated in part by the emergence in the Internet of applications for audio- and video-conferencing such as vat, the visual audio tool, which adapts its playback time to the delay in the network. The model of predictive service differed from service models discussed in previous research in that new connections would not get an agreement from the network that the connection's deterministic or statistical requirements about delay and packet loss rates will

be met. The proposed admissions control procedure for predictive service is based not on the flow specifications of the admitted connections, but on the measurements of past traffic in the class. The admissions control procedure for the class assumes that the past traffic in the class can be used as a predictor of future traffic. If this prediction turns out to be incorrect and the network becomes oversubscribed, then either the delay for the various connections increases and is accommodated by the adaptive playback time, or the service becomes sufficiently unacceptable that some users receiving the realtime traffic decide to terminate the connection. This model of predictive service was in part a response to the emergence of new applications in the internet.

Like the model of predictive service, it seems likely that additional service models will emerge to meet the needs of emerging applications. One possible new application is variable-bit-rate video that has a congestion control procedure that adapts the source transmission rate as network conditions change. As one example of this emerging class, Wakeman and Crowcroft [WC92] propose a combined admission and congestion control scheme for variable-bit-rate video. In this scheme the host determines the state of the network, determines whether to introduce an additional variable-bit-rate video flow, and then adjusts the source coding rate to share the bandwidth in a fair manner. In [G93], Garrett proposes a layered coding scheme for variable-bit-rate video using drop-preference, where the video traffic is partitioned into two priority layers to protect the higher-priority packets from loss. The proportion of video traffic placed in each layer is determined by feedback from the network. Note that this is different from simply using drop-preference to indicate which packets should preferentially be dropped by the gateway; the proposal in [G93] explicitly assumes that some small but non-zero rate of packet drops at the gateway is acceptable, and that the video coder can respond to feedback from the network about the level of congestion.

The model of variable-bit-rate video with adaptive coding could be easily accomodated in a network with link-sharing, where each gateway allocates a certain (possibly dynamic) link-sharing bandwidth to a class of variable-bit-rate video connections. In the absence of congestion, the video traffic could use as much bandwidth as desired; in the presence of congestion, the bandwidth of the video class would be controlled, and some arriving video packets might be dropped at the gateway until the adaptive coding can reduce the congestion. For a class of video traffic where the users require a certain minimum bandwidth, but users can also use a range of bandwidths above the minimum bandwidth, an admissions control procedure might be used in addition to the end-to-end congestion control. For a class of video traffic where the user would rather receive 1 frame each n seconds rather that receive no frames at all, (for example, perhaps for talking-heads video accompanying an audioconference), then an admissions control procedure might not be required.

Because link-sharing can be used to limit the bandwidth of traffic classes during times of congestion, link-sharing can be used to isolate traffic classes from each other. Thus link-sharing can be an important mechanism in accomodating emerging service models in the Internet.

Another example of the need for flexibility in the specification of service models is the proposed division of realtime traffic into predictive service connections, which have adaptive playback times and are tolerant of packet loss, and guaranteed service connections, which have fixed playback times and are intolerant of packet loss [CSZ92, SCZ93]. Thus in [SCZ93] the guaranteed service is described as providing a 'perfectly reliable upper bound on the maximum delay of each packet'. Another class of traffic that could emerge could be realtime connections that have some tolerance of packet loss, but that either do not have adaptive playback times or are not tolerant of persistent delay. Such a class of traffic could be provided by a class of traffic with statistical multiplexing, where the admissions control procedure is based on the flow specifications of the admitted connections as well as on the measurements of past traffic. Because of the statistical multiplexing, this service would only be for connections that could tolerate some small fraction of packet drops. Because the admissions control procedure would take into account the flow specifications of the admitted connections, the admissions control procedure would be more conservative that an admissions control procedure intended for adaptive service classes [CSZ92]. Whether this service model will correspond to a significant future need in the Internet is difficult to predict.

It is necessary to consider possible conflicts in a network where gateways have both a general scheduler to accommodate the needs of real-time traffic and link-sharing mechanisms to share traffic on individual links between various agencies, protocol families, or traffic types. In the paper we use the term *realtime traffic* to refer to traffic that has a playback time, where packets that arrive at the receiver after that time are discarded; realtime traffic most likely will use some admissions control procedure to establish a new connection.[4] By *non-realtime traffic* we mean traffic where low delay might be desirable, but the receiver waits until packets are received. In general, non-realtime traffic will have

[4]Current audio and video traffic on the Mbone has an informal, unenforced admissions control procedure that depends on rough consensus between the users of the Mbone.

no admissions controls, no guarantees about delay or about packet drop rates, and a reliance on end-to-end protocols for congestion control, as in the current Internet. One danger in the current research on providing realtime services is to discount the needs of non-realtime traffic in the Internet.

One possibility for a scheduler at the gateway would be to consider the needs of the realtime traffic first, and to schedule the non-realtime traffic after the needs of the realtime traffic had been met, without having the scheduler enforce any bandwidth limitations on the realtime traffic over any time scale [SCZ93]. After the needs of the realtime traffic had been met, link-sharing mechanisms would be used to share the remaining bandwidth among the non-realtime classes. We argue, however, that this type of approach to link-sharing is not sufficient. Either it could lead to starvation of the non-real-time traffic over substantial periods of time, or it restricts the types of real-time traffic that could be accomodated by the network, placing undue reliance on the admissions control procedure for realtime traffic and on the policing of realtime traffic at the edge of the network. For example, such an approach might not be the best mechanism to accomodate drop-preference video traffic that adapts its source coding to adapt to network conditions.

This section proposes that link-sharing explicitly enforced at the gateway for both realtime and nonrealtime traffic can protect non-realtime traffic from transient starvation, satisfy the needs of realtime traffic, and give the network the flexibility to accommodate new realtime applications.

One example of possible conflict at the gateway is between the realtime service goals of predictive service traffic and the link-sharing goals. The commitment of predictive service is that the network will use admissions control procedures based on the past traffic of the network to control the admissions of predictive service connections. When these predictions from past traffic are reliable, the predictive service packets should be delivered with appropriately low delay. It is not possible, given such an admissions control procedure, to make quantitative commitments about the level of service when the predictions from past traffic turn out to be unreliable. One option would be for a gateway, in this case, to serve as many predictive service packets as possible (given that other realtime service commitments are met). Another option would be for the gateway to restrict the predictive service traffic as necessary to meet the link-sharing goals, including the link-sharing goal of allocating bandwidth for non-realtime traffic over some time period.

Thus when the predictive service class's predictions from past traffic has been incorrect, there can be a conflict between the predictive service goal of imperfectly reliable delay bounds and some of the link-sharing goals. The bandwidth of predictive service traffic is in part controlled by the admissions control procedure; the effectiveness of this control depends on the rate at which predictive service connections start and stop, for the class at that link. For a high-bandwidth link with frequent changes in the number of predictive service connections, the predictive service admissions control procedure could be sufficient to control the bandwidth of predictive service traffic over appropriate time scales. First, the assumption of the predictive service admissions control procedure that past traffic is a reliable guideline for future traffic is more effective when there are a large number of predictive service connections. Second, for those times where the predictive service admissions control procedure is overoptimistic, this could be corrected by simply waiting a short time until some of the predictive service connections terminate.

However, for a link of moderate bandwidth, where a moderate number of possibly long-lived predictive service connections could use a substantial fraction of the link bandwidth, the predictive service class's admissions control procedure would be less effective in protecting the link-sharing goals. Given the moderate number of predictive service connections, the assumption that past traffic is a reliable predictor of future traffic will not always be correct. When the prediction is incorrect, the predictive service traffic could exceed the desired link-sharing bandwidth. In the presence of congestion, this could be corrected either by waiting until some of the (possible long-lived) predictive service connections terminate, or by having the scheduler restrict predictive service traffic over an appropriate time interval. Neither of these options violate the commitment of the predictive service goal, which is to provide reliable delay bounds *contingent on* the assumption that past traffic has been a reliable indicator of future traffic. The second option, however, protects the quantitative link-sharing goals as well as the predictive service goals. If this second option is used, the predictive service admissions control procedure should only admit new connections if traffic measurements indicate that the delay would have been acceptable even if the class as a whole had been restricted to its allocated bandwidth. (An easy implementation of this would simply be not to admit new predictive service connections when the predictive service class has been exceeding its allocated bandwidth over relevant time intervals.)

Thus, the enforcement of link-sharing at the gateway does not have to result in unacceptable service for real-time traffic. The link-sharing guidelines only apply over a specified time interval, and do not imply that a class is restricted to its link-sharing bandwidth over arbitrarily-

small time periods. A realtime class will never be regulated by the link-sharing scheduler if the general scheduler, the admissions control procedure, and the policing of admitted connections at the edge of the network can together be relied upon to ensure that the realtime class doesn't exceed its link-sharing bandwidth over relevant time periods.

7 Related work

This section discusses some other approaches to implementing link-sharing in networks.

The approach to link-sharing in this paper is derived from an approach proposed by Van Jacobson, along with others on the End-to-end Research Group [CJ91]. In that proposal, a single set of mechanisms is proposed to implement link-sharing and realtime services. The mechanisms proposed are a *classifier* to classify arriving packets to the appropriate class, an *estimator* to estimate the bandwidth recently used by a class, a *selector* to determine the order in which packets from the various classes will be sent, and a *delayer* or *overlimit action* to schedule traffic from classes that have exceeded their link-sharing limits and are contributing to congestion. The classifier and the estimator are separate mechanisms whose functions can be fairly clearly defined. The selector corresponds to the general scheduler defined in this paper, and the delayer or overlimit action corresponds to the link-sharing scheduler in this paper.

The link-sharing scheme proposed in [SCZ93] has been discussed elsewhere in this paper. The approach in [SCZ93], after defining a service model including both quality of service commitment to individual flows and link-sharing commitments to collective entities, is to define a precedence ordering for the various service goals, including the link-sharing goals, and to use this precedence ordering to specify which commitments should be satisfied first when commitments conflict. Our paper differs from [SCZ93] not in the description of the service model but in the service architecture that results from the consideration of all of the goals. The goals in [SCZ93] include guaranteed real-time service, predictive real-time service, several classes of as-soon-as-possible service for non-real-time traffic, and hierarchical link-sharing.

The goal of hierarchical link-sharing described in [SCZ93] is defined as approximating, as close as possible, the bandwidth shares provided by an idealized fluid model of instantaneous link-sharing. There are two significant differences from our approach to link-sharing and the approach in [SCZ93].

First, the link-sharing goal in [SCZ93] is restricted to providing link-sharing between entities (or classes) of the same priority. The link-sharing goal of approximating an idealized fluid model does not allow for explicit link-sharing between two classes with different service priorities, such as between a realtime and a non-realtime class, or between a telnet and an ftp class of traffic. Their model could perhaps be extended include such a possibility, but this is not discussed in the paper.

Second, the link-sharing in [SCZ93] takes into account the bandwidth used by realtime traffic, but the link-sharing algorithm does not affect the scheduling of the realtime traffic. From [SCZ93]: "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 of 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." [SCZ93]

Section 6 of our paper discusses how the explicit enforcement of link-sharing goals for realtime traffic suits realtime traffic that can accomodate a small but nonzero rate of packet drops, such as real-time traffic with drop-preference and/or rate-adaptive congestion control algorithms. Section 6 of our paper also presents our argument that this explicit enforcement of link-sharing goals for realtime traffic contributes to the flexibility of the resource management architecture, and does not violate the goals of guaranteed or predictive real-time service. There is discussion in [SCZ93] of the possibility of providing for dropping a small percentage of "expendable" realtime packets from hierarchically-encoded video when realtime utilization targets are in danger of being violated, but there is no discussion of how this could be accomplished. It is our contention that such issues should be considered within the context of hierarchical link-sharing.

[Further discussion might have to wait until I have thought some more about the new version of their paper.]

[The discussion of Fair Share is based on a draft version of a paper.] The Fair Share proposal outlined in [S93] consists of a set of mechanisms for allocating a network resource such as link bandwidth among competing traffic flows. The Fair Share proposal involves bandwidth reservations for those traffic flows that have explicit throughput requirements, along with provisions for sharing unused capacity among the contending flows. We examine the link-sharing aspect of the Fair Share proposal in some detail because it is substantially different from the proposal in our paper.

Fair Share assumes a hierarchical structure of classes or *flow sets* for an output link. Fair Share provides guaranteed throughput for a class by reserving bandwidth for that class, and by then controlling the rate

at which traffic from that class is allowed on the link. Statically-configured classes have infinite lifetimes, while dynamically-configured classes have finite lifetimes. The Fair Share mechanism periodically computes the permitted rate for each class at regular intervals (*rate computation intervals*), where the permitted rate depends on the bandwidth guaranteed to the class, the past permitted rate for that class, and the past measured link utilization. For each packet arriving at the gateway for a particular class, Fair Share decides whether or not to accept that packet, depending on the permitted rate computed for that class and on the size of the current queues.

In the Fair Share scheme, each class has two types of packets and two types of queues, a guaranteed queue and an excess queue. In each *rate enforcement interval* the gateway first assigns arriving packets for the class to the guaranteed queue for that class, then assigns arriving packets to the excess queue for that class, and then discards arriving packets. At the beginning of each rate enforcement interval, Fair Share discards any excess packets from the previous rate enforcement interval. The queues for classes are served in round-robin order. The only use of priority among the various classes is in the acceptance, rejection, or elimination of classes.

One significant difference between our proposal and the Fair Share proposal is that Fair Share does not have a provision for priorities of service among various traffic classes. Thus there is no mechanism for providing different levels of service for delay-sensitive and delay-insensitive traffic.

The implementation of link-sharing in the Fair Share scheme requires the computation of permitted rates for each class. Because the permitted rate computed for each class depends on the past permitted rate for that class and on the measured past bandwidth for the link as a whole, but not on the measured past bandwidth used by that particular class, the convergence behavior of this system is somewhat complex, and could be slow in some cases. It seems possible that the use of rate enforcement intervals in Fair Share could contribute to periodic traffic patterns in the network. As discussed in the paper, the length of the rate computation interval also affects the stability of the traffic rate control.

The intention of Fair Share seems to be that each real-time conference would have its own class, with its own allocated bandwidth. There is little provision for statistical multiplexing, or for a class of real-time connections with adaptive playback times. This lack of provision for statistical multiplexing is not necessarily inherent in the Fair Share scheme; however, because the Fair Share gateway does not monitor the bandwidth or performance of an individual class, the Fair Share scheme seems unsuited to providing for a predictive traffic class of connections with adaptive playback times as in [CSZ92].

8 Conclusions and future work

One of the strengths of the link-sharing framework proposed in this paper is that it combines link-sharing provisions with other gateway provisions for priority (or other gateway scheduling mechanisms) for delay-sensitive traffic. The proposed link-sharing framework allows for priority-based or other scheduling algorithms at the gateway for realtime traffic, while "folding in" provisions to protect the link-sharing behavior at the gateway. We argue that link-sharing goals should be considered at the same time that real-time service goals at the gateway are considered. This can lead to more efficient and productive implementations of both services.

The incorporation of link-sharing mechanisms along with the provision of realtime services can simplify and add robustness to the provision of realtime services in the Internet. Link-sharing mechanisms at the gateways add considerably to the flexibility of the Internet in accomodating emerging real-time applications, or in accomodating emerging transport protocols whose response to congestion is more or less conservative that that of TCP.

It is not necessarily possible at this time to fully anticipate the service requirements of emerging real-time applications on the Internet. Thus, in addition to continuing the discussion on what resource management services we anticipate will be required from the future Internet, we emphasize that our understanding of these resource management services is not necessarily stable. A key requirement of the resource management architecture will be the flexibility to satisfy the requirements of emerging real-time applications. Link-sharing enforced at the gateway is one mechanism for contributing to this flexibility.

This paper addresses only one aspect of resource management issues required at the gateway, that of link-sharing. Related issues include set-up protocols and flow specifications for realtime applications, admissions control procedures at the gateway, and questions about scheduling algorithms at the gateway. Many of these issues are being addressed by other groups of researchers, and some of these issues are being addressed in more detail at LBL. [I should say more. This paper is a draft paper, and is definitely not in its final stage.]

9 Acknowledgements

Thanks go to Dave Clark, Sugih Jamin, Scott Shenker, Lixia Zhang, and to the members of the End-to-end Research Group for many discussions of link-sharing and of other issues of resource management. Thanks also to go Steven McCanne, who has done much of the work in modifying and maintaining our simulator.

This paper resulted from joint work with Van Jacobson. [He isn't listed as a co-author at the moment in part because he hasn't read all of the paper.]

[In general, I intend to add more references, citing some of the more relevant articles from the body of research on all of this.]

References

[CD92] Casner, S., and Deering, S., "First IETF Internet Audiocast", ConneXions, V.6 N.6, June 1992, p.10-17.

[CJ91] Clark, D.D. and Jacobson, V., "Flexible and Efficient Resource Management for Datagram Networks", unpublished draft, April 1991.

[CSZ92] Clark, D.D., Shenker, S., and Zhang, L., "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism", SIGCOMM "92, August 1992, p. 14-26.

[FJ93] Floyd, S., and Jacobson, V., *Random Early Detection Gateways for Congestion Avoidance*, available by anonymous ftp from ftp.ee.lbl.gov:early*.ps.Z. To appear in IEEE/ACM Transactions on Networking.

[G93] Garrett, M., Contributions Toward Real-Time
Services on Packet Switched Networks, CU/CTR/TR 340-93-20, Columbia University, 1993.

[J88] Jacobson, V., *Congestion Avoidance and Control*, Proceedings of SIGCOMM '88, August 1988, pp. 314-329.

[JSZC92] Jamin, Shenker, S., Zhang, L., and Clark, D., "An Admission Control Algorithm for Predictive Real-time Service", Proceedings of the Thirs Internationsl Workship on Networking and Operating System Support for Digital Audio and Video, 1992.

[SCZ93] Shenker, S., Clark, D.D., and Zhang, L., "A Scheduling Service Model and a Scheduling Architecture for an Integrated Services Packet Network", preprint, 1993.

[S93] Steenstrup, M., "Fair Share for Resource Allocation", preprint, 1993.

[WC92] Wakeman, I., and Crowcroft, J., "A Combined Admission and Congestion Control Scheme for Variable Bit Rate Video", October 1992.

[Y84] Young, P., Recursive Estimation and Time-Series Analysis, Springer-Verlag, 1984, pp. 60-65.

A Implementation of the estimator

The estimator determines the limit status of the classes in the class structure. This appendix describes the implementation of the estimator in our simulator; this is only one of several methods that could be used to implement the estimator. One requirement for the estimator is that it should be efficient to implement.

There are two related questions to consider in specifying an estimator. The first question concerns the time interval over which the limit status of a class should be computed, or using slightly different terminology, the time constant of the estimator. The time constant of the estimator is an explicit design parameter at the gateway. The second question concerns how frequently the limit status of a class should be computed.

In our simulator, the gateway recomputes the limit status for a class (and its ancestor classes) after the gateway transmits a packet from that class. Our estimator uses an exponential weighted moving average (EWMA), (as is used in 4.3BSD TCP to compute the mean of the round trip delay [J88]). This estimator looks at recent inter-packet departure times, with a decaying weight for the more distant packets, and indirectly computes the mean of the inter-packet departure time, or (in the reciprocal) the mean of the packet rate.

To be somewhat more precise, let s be the size of the recently-transmitted packet in bytes, let b be the link-sharing bandwidth allocated to the class in bytes per second, and let t be the measured interdeparture time between the packet that was just transmitted and the previous packet transmitted from this class, as shown in Figure 14. If the gateway sends packets of size s from the class at precisely the link-sharing bandwidth b allocated to the class, then the interdeparture time between

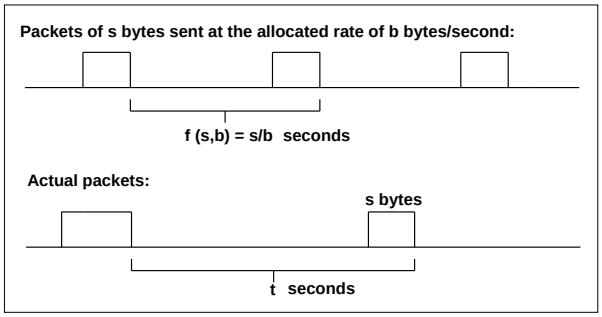

Figure 14: Variables for the computation of the limit status of a class.

successive packets would be

$$f(s,b) = s/b$$

seconds. Let

$$diff = t - f(s,b)$$

be the discrepancy between the actual interdeparture time and the "allocated" interdeparture time for that class for packets of that size. Note that *diff* is negative when the class is exceeding its link-sharing bandwidth and positive otherwise. Our simulator computes *avg*, the exponential weighted moving average of the *diff* variable, using the following equation:

$$avg \leftarrow (1-w)avg + w * diff.$$

With properly scaled versions of the parameters, and with the weight w chosen as a negative power of two, *avg* can be computed with one shift and two add instructions [J88]. In computing *diff*, the function $f(s,b)$ could be computed explicitly, or could be determined by using the packet size s as an index into an array for that class.

The weight w determines the time constant of the estimator. If the sending rate for a class suddenly changes, causing the computed value of the *diff* variable to change from one value to another, then it takes $-1/\ln(1-w)$ packets to be sent from that class before the computed estimate *avg* moves 63% of the way from the old value of *diff* to the new value [Y84]. This corresponds to a *time constant* of roughly

$$\frac{-s}{b\ln(1-w)}$$

seconds for a class sending s-byte packets at close to the class's allocated link-sharing bandwidth of b bytes/second.

With this method for implementing the estimator, a class that has used only a small fraction of its allocated bandwidth for an extended period of time could have a large value for the parameter avg. A previously-idle class with a link-sharing allocation of b bytes/second and a value of A for *avg* could send n back-to-back s-byte packets before being estimated as overlimit, for

$$n < \frac{\log(\frac{A}{(s/b-s/l)}+1)}{-\log(1-w)}. \qquad (1)$$

[This uses the fact that for a class sending back-to-back packets, the measured interdeparture time would be s/l seconds, for l the link bandwidth in bytes per second. With back-to-back packets, the computed (negative) value for *diff* would be $s/l - s/b$ seconds. After n back-to-back packets, the value for *avg* would be

$$(1-w)^n A + \sum_{i=0}^{n-1}(1-w)^i w(s/l - s/b)$$

$$= (1-w)^n A + (1-(1-w)^n)(s/l - s/b).$$

The value for *avg* will still be positive as long as

$$A > (s/b - s/l)((1-w)^{-n} - 1).$$

This holds for values of n that satisfy Equation 1 above.]

By limiting the maximum value for the variable *avg*, the estimator can limit the number of back-to-back packets that can be sent from a previously idle class before the class is estimated as overlimit.

Note that our estimator does not actually estimate the bandwidth used by a class. The estimator is fairly accurate in estimating whether a class is over or under its limit, but the exact value of *avg* computed by the estimator can be sensitive to things such as the packet sizes used by the class.

For Formal link-sharing, and for leaf classes in the Ancestors-Only and the Top-Level link-sharing, the scheduler needs to know whether or not a class is *overlimit*. In this case the estimator uses the actual link-sharing allocation b of the class in computing $f(s/b)$, the "allocated" interdeparture time between packets. In contrast, for interior classes in the Ancestors-Only and the Top-Level versions of link-sharing, the scheduler needs to know whether or not the class is *underlimit*. For this, the estimator uses a bandwidth b' that is slightly less than the link-sharing bandwidth b allocated to the class in computing the "allocated" interdeparture time $f(s,b')$. For example, in computing whether or not the root class is underlimit, the estimator uses a bandwidth b' somewhat less than the actual bandwidth of the link in computing the "allocated" interdeparture time $f(s,b')$.

For a nonregulated class, the gateway uses the computed value of avg to set the time-to-send field for the class. If avg is positive, then the time-to-send field is set to zero, indicating that the class is under its limit. If

avg is negative, then the time-to-send field for the class is set to a time x seconds ahead of the current time, for

$$x = -avg\frac{1-w}{w} + f(s,b), \qquad (2)$$

and for s the size of the packet just transmitted from the class. If the gateway waits x seconds before sending another packet from the class, then the class will no longer be over its limit. Note that for an unregulated class, the time-to-send field only indicates whether or not the class is currently over or under its limit. The time-to-send field does not in any way guarantee that a nonregulated class will receive its link-sharing allocation. The actual bandwidth received by a nonregulated class is determined by the general scheduler.

For a regulated class, the link-sharing scheduler sets the time-to-send field for the class to the earliest time that the class should next be able to send a packet. This is described in Section C.

An alternate implementation for the estimator would be for the gateway every t seconds to recompute the limit status for each class over the last T seconds. As long as $t \ll T$, this should be adequate. With this implementation, the accuracy of the gateway in satisfying the link-sharing goals is limited by the ratio between T and t. The value for T is determined by the desired time intervals over which the link-sharing goals should be enforced. Given T, decreasing t increases the accuracy of the gateway in satisfying the link-sharing goals.

B Implementation of the general scheduler

The general scheduler schedules packets from unregulated classes at the gateway. This section describes the implementation of the general scheduler in our simulator.

In our simulator, a separate queue is maintained at the gateway for each class associated with that output link. After each packet is transmitted on the output link, the general scheduler decides which class can next send a packet on the link. The general scheduler schedules packets from higher-priority classes first. Within classes of the same priority, the general scheduler uses a variant of weighted round-robin, with weights proportional to the bandwidth allocations of the classes. The weights determine the number of bytes that a class is allowed to send at each round. When a class sends more than its allocated number of bytes (because packets aren't broken into byte-sized pieces), that class's byte-allocation for the following round is correspondingly reduced.

The use of weighted round-robin to service classes of the same priority level serves two functions. The first function is to ensure that each priority-one class receives its allocated bandwidth even over fairly small time intervals. If at most half of the link bandwidth is allocated to priority-one classes, then each priority-one class with sufficient demand is guaranteed to receive at least its allocated bandwidth in each round of the round-robin.

The second function of the weighted round-robin is to ensure that in the absence of link-sharing mechanisms, bandwidth is distributed to classes of the same priority in proportion to the bandwidth allocations of those classes. This results in "excess" bandwidth being distributed by the general scheduler in proportion to the bandwidth allocations of the leaf classes at that priority level.

Before the general scheduler transmits a packet from a class, the scheduler checks that the time-to-send field for the class is less than the current time. The scheduler checks limit status of a class simply by comparing the class's time-to-send field with the current time. If the time-to-send field is zero, then the class is at-limit or underlimit, and the general scheduler is allowed to transmit a packet from that class.. If the time-to-send field is nonzero but less than the current time, then the class might be overlimit, but the general scheduler is still allowed to transmit a packet from that class.

If the time-to-send field for a class is greater than the current time, then the class is overlimit. In this case, the general scheduler can only send a packet from that class if permitted by the link-sharing guidelines. (For example, with Ancestor-Only link-sharing, if the time-to-send field for a class is greater than the current time, then the general scheduler can only send a packet from that class if there is an underlimit ancestor class.

An essential issue for any proposal for a general scheduler is that the scheduler should lend itself to efficient implementations. However, this paper does not discuss such implementation issues.[5]

C Implementation of the link-sharing scheduler

The link-sharing scheduler controls the scheduling of packets from regulated classes. This section describes the implementation of the link-sharing scheduler in our simulator. In our simulator the link-sharing scheduler, working in concert with the general scheduler, effectively rate-limits regulated classes to their allocated link-sharing bandwidth.

Our simulator uses two different methods for rate-limiting a regulated class. The two methods give similar results, but depending on the circumstances, one or

[5]The efficiency of our scheduler implementation has been demonstrated through experiments in the DARTnet test bed.

the other method might be preferred for reasons of efficiency. We describe only one of the methods in this section.

In our simulator's implementation, the link-sharing scheduler sets the time-to-send field for a regulated class to $f(s,b) = s/b$ seconds ahead of the current time, where the most-recently-sent packet, transmitted at the current time, contained s bytes, and the class has a link-sharing allocation of b bytes/second. The result is that the general scheduler considers the class as overlimit until the time indicated in the time-to-send field; at that time the general scheduler is allowed to send a packet from that class regardless of the limit status of ancestor classes. If the class is still overlimit after a packet is sent (as indicated by the *avg* variable maintained for that class), then the time-to-send field is again set to $f(s,b) = s/b$ seconds ahead of the current time.

Notice that in our simulator, for a regulated class the exact scheduling of packets from the regulated class is still determined by the general scheduler. For a high-priority class, with a general scheduler such as ours that uses priority, the general scheduler is likely to send a packet from a regulated class soon after the time indicated in the time-to-send field. For a lower-priority class, the general scheduler could be delayed somewhat longer before sending a packet from a regulated class. If this happens frequently, then the previously-overlimit regulated class will soon no longer be overlimit, and will no longer need to be regulated by the link-sharing scheduler. At that point, the time-to-send field for that class will be reset to zero.