Link-sharing and Resource Management Models for Packet Networks

Sally Floyd and Van Jacobson
To appear in IEEE/ACM Transactions on Networking, Vol. 3 No. 4, August 1995

Abstract—

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

While it is not possible to completely predict the requirements that might evolve in the Internet over the next decade, we argue that controlled link-sharing is an essential component that can provide gateways with the flexibility to accommodate emerging applications and network protocols.

1 Introduction

Requirements for resource management in the Internet include both services for real-time traffic and link-sharing services. Real-time traffic is characterized by a (fixed or adaptive) playback time at the receiver; real-time packets arriving at the receiver after the playback time are discarded. In a congested network, resource management mechanisms are required at the gateway to meet realtime traffic requirements for controlled delay and limited packet drops. While there has been an abundance of research about the needs of real-time traffic, link-sharing services have received somewhat less attention in the research community.

The approach to controlled link-sharing described in this paper has evolved in the context of the Internet. Because the Internet is decentralized in nature, composed of multiple administrative domains with a wide range of resource limitations, the control of Internet resources involves local decisions on usage as well as considerations of per-connection end-to-end requirements. One function of link-sharing mechanisms is to enable gateways to control the distribution of bandwidth on local links in response to purely local needs. By allowing isolation between real-time and best-effort traffic in cooperation with packet scheduling algorithms that give priority to the real-time traffic, controlled link-sharing can also be a key component in enabling the deployment of priority-based packet scheduling algorithms designed to meet the end-to-end service requirements of real-time traffic.

One requirement for link-sharing is to share bandwidth on a link between multiple organizations, where each organization wants to receive a guaranteed share of the link bandwidth during congestion, but where bandwidth that is not being used by one organization should be available to other organizations sharing the link. Examples range from the multiple agencies that share the Trans-Atlantic FAT pipe and each pay a fixed share of the costs [WGCJF94] to individuals who share a single ISDN line. Another requirement for link-sharing is to share bandwidth on a link between different protocol families (e.g., IP and SNA), 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, ftp, or real-time audio and video.

We believe that the needs met by link-sharing are fundamental, given the presence of congestion. In particular, we believe that the need to share links between multiple organizations is not a transient stage that will disappear with the full commercialization of the Internet (until or unless congestion itself disappears). The hierarchical structure of organizations is not a transient phenomena, and, given the availability of an appropriate link-sharing framework, is likely to be reflected in a continued desire for controlled link-sharing of local resources such as network bandwidth. Additional needs met by controlled link-sharing, discussed in more detail later in the paper, include the ongoing need to accommodate new services, and the need to control traffic aggregation in order to realize the advantages of sharing between connections using compatible congestion control mechanisms.

The various requirements for link-sharing, taken together with requirements for realtime services, 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).

Link-sharing services and real-time services involve si-

S. Floyd and V. Jacobson are both with the Network Research Group, Lawrence Berkeley Laboratory, Berkeley CA (email: floyd@ee.lbl.gov and van@ee.lbl.gov). This work was supported by the Director, Office of Energy Research, Scientific Computing Staff, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098 and by ARPA/CSTO.

This is a revised version of an earlier draft, dated September 1993, that was made available over the Internet. Copyright ©1995 by IEEE.

multaneous sets of constraints to be satisfied at the gateway. This paper addresses the interaction between real-time services and link-sharing services at the gateway. A key contribution of this paper is an investigation of ways that link-sharing can be incorporated into more general scheduling frameworks such as priority-based scheduling. This paper proposes that link-sharing explicitly enforced at the gateway can prevent starvation of lower-priority traffic while still satisfying the needs of higher-priority traffic, and give the network the flexibility to accommodate new real-time applications.

As an example of the interaction between real-time services and link-sharing, consider a link shared between two classes of traffic, a real-time class and a bulk-data class. For the purposes of the link-sharing algorithms, we make a conceptual distinction between a *general scheduler* and a *link-sharing scheduler*. In the absence of congestion, the gateway could use whatever general scheduler seemed most appropriate, ranging from a priority-based to a round-robin scheduler. However, in the presence of congestion the gateway might determine that one of the two classes was using more than its allocated share of the link bandwidth, and invoke the link-sharing scheduler to rate-limit the *overlimit* class to its allocated bandwidth. This paper does not attempt to outline a complete packet-scheduling algorithm; we instead are proposing a mechanism for incorporating controlled link-sharing into the packet scheduling framework.

However, instead of implementing real-time and link-sharing services with separate pieces of code, it is preferable to use an integrated set of mechanisms for real-time and link-sharing services. Identifying a set of low-level mechanisms that can implement these services, and separating the low-level mechanisms from higher-level policy, gives a flexible resource management framework that allows evolution. Instead of outlining a complete service model for the Internet, we explore a set of low-level mechanisms that can be used to efficiently support a range of real-time and link-sharing services. Flexibility in the resource management framework is particularly important because it is not easy to fully anticipate the service requirements of emerging applications on the Internet. Controlled Link-sharing makes a key contribution to this flexibility.

This paper focuses on the role of link-sharing in the resource management framework. As an example of the flexibility afforded by link-sharing, consider the current need in the Internet for some mechanism to protect data traffic from the growing volume of real-time Mbone traffic [E94], as well as the need to protect the real-time traffic from the delays caused by competing data traffic. Given an environment with limited bandwidth, fully meeting the needs of real-time traffic requires a suite of real-time services including flow specifications for the real-time applications, a set-up procedure such as RSVP [ZDESZ93], and admissions control procedures to control the number of admitted real-time connections, in addition to appropriate scheduling mechanisms at the gateway. However, by guaranteeing that data and real-time traffic each receive a share of the link bandwidth over relevant time intervals, link-sharing mechanisms can protect both Mbone and data traffic, in the aggregate, even in the absence of a full suite of real-time services.

We are not suggesting that each link in the Internet requires a separate traffic class for each agency, protocol family, or traffic type traversing that link. For example, organization-based link sharing might be needed for a link such as the Trans-Atlantic FAT pipe while another link has no need for organization-based link sharing. Because different links in a heterogeneous Internet will have different link-sharing structures, two connections that are aggregated into one class on one link might be in separate classes on the following link. One benefit of the link-sharing framework proposed in this paper is that it acknowledges the decentralized nature of the Internet, and allows some local control of bandwidth distribution.

The link-sharing goals, described in more detail in Section 2, are quite modest. The link-sharing mechanisms take the minimum action required to ensure that classes receive their allocated link-sharing bandwidth over the relevant time interval. The link-sharing mechanisms in this paper do not attempt to provide congestion control within a "leaf" class, 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. These issues are determined by the general scheduler used at the gateway. The link-sharing mechanisms do not, by themselves, attempt to implement arbitrary scheduling policies.

A hierarchical link-sharing structure can be used to specify guidelines for the distribution of "excess" bandwidth. While one could imagine complex requirements in terms of exactly how "extra" bandwidth is distributed or what fraction of bandwidth each class requires over a range of time intervals, in this paper we restrict our attention to those fairly straightforward requirements that can be expressed by these hierarchical structures. As section 2 notes, while the distribution of "extra" bandwidth beyond the constraints imposed by the hierarchical link-sharing structure should not be arbitrary, this distribution is a function of the general scheduler, and is not addressed in this paper.

The approach to link-sharing in our paper is based on the hierarchical class-based resource management proposed initially by Van Jacobson [CJ91]. This approach, now referred to as class-based queueing (CBQ), outlines a set of flexible, efficiently-implemented gateway mechanisms that can meet a range of service and link-sharing requirements. Appendix A discusses the implementation of CBQ in our simulator, and gives a pointer to publically-available distributions of CBQ implementations.

Our paper is in the context of a continuing discussion about resource management in the Internet that involves contributions from many people. As examples, see [BCS94, CJ91, CSZ92, SCZ93, FBZ94]. The form of this paper was motivated in part as a response to the scheduling architecture proposed in [SCZ93]. We are in substantial agreement with much in [SCZ93]; Section 8 of this paper elaborates on

a disagreement over the relationship between link-sharing services and real-time services. Thus, the framework for this paper borrows heavily from the framework in [SCZ93].

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 real-time traffic. Section 7 discusses link-sharing in terms of isolation and sharing between traffic. Section 8 compares the link-sharing framework discussed in this paper with related work. Section 9 gives conclusions and discusses future work.

2 The link-sharing goals

This section gives a general discussion of the link-sharing goals. The requirements for link-sharing are essentially the same whether the link-sharing is between service classes, organizations, protocol families, or traffic types. We argue that a single set of mechanisms for link-sharing should be implemented and carefully coordinated with any additional mechanisms for providing real-time service.

The link-sharing structure specifies the desired policy in terms of the division of bandwidth for a particular link in times of congestion. For example, for the link-sharing structure in Figure 1 the link is shared by a number of real-time and non-real-time 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. 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 NNTP and FAX as well as mail traffic.

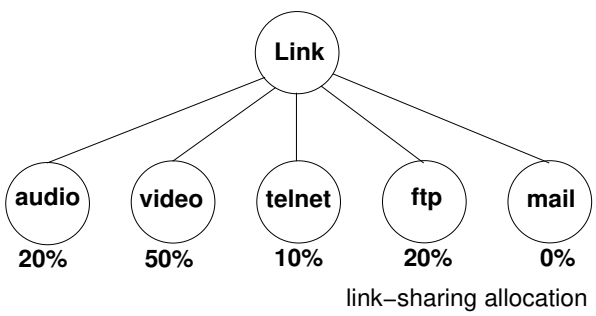

Figure 1: Link-sharing between service classes.

For a flat link-sharing structure such as 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 percentage of the overall link bandwidth). These link-sharing allocations could be either static (permanently assigned by the network administrator) or dynamic (varying in response to current conditions on the network, according to some predetermined algorithm). 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. As discussed later in the paper, this is determined by the time constant used in estimating the past bandwidth used by each class. For example, in Figure 1 it might be considered unacceptable if the **telnet** and **ftp** classes were denied service for minutes at a time. On the other hand, fine-grained scheduling to ensure that the **telnet** and **ftp** classes each receive their allocated link-sharing bandwidth over arbitrarily-small time intervals is not required. Priority-based scheduling can be used to reduce delay for the real-time traffic, while the link-sharing mechanisms prevent starvation of the ftp traffic over longer time intervals.

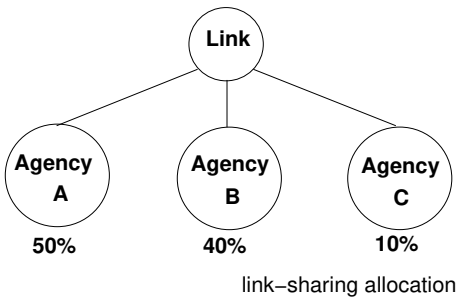

Figure 2: Link-sharing between multiple agencies or protocol families.

A secondary link-sharing goal is that when some class is not using its allocated bandwidth, 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 determined by the other scheduling mechanisms used at the gateway, and is not specified by the link-sharing structure. For example, consider link-sharing between organizations or 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 scheduling mechanisms could distribute 'excess' bandwidth in a way that takes into account the relative link-sharing allocations of those

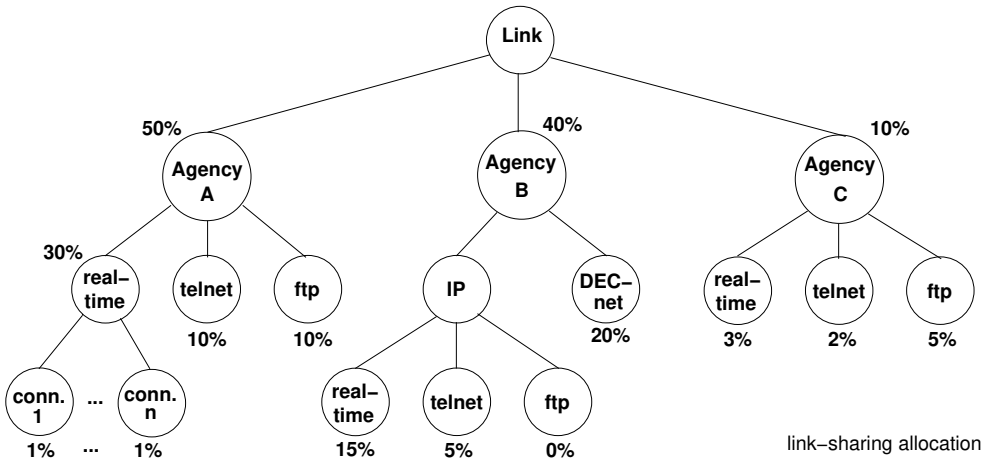

Figure 3: A hierarchical link-sharing structure.

entities.

Multiple link-sharing constraints at a gateway can be expressed by a hierarchical link-sharing structure such as in Figure 3. The link-sharing structure in Figure 3 illustrates link-sharing between organizations, between protocol families, between service classes, and between individual connections within a service class; this is not meant to imply that all link-sharing structures at all links should include all of these forms of link-sharing. 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 real-time 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 appropriate 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. □

The implementation of the first link-sharing goal is discussed in detail in this paper. As mentioned in the Introduction, the first link-sharing goal is limited by the constraints of the hierarchical class structure.

The second link-sharing goal concerning the further distribution of excess bandwidth is not addressed in this paper. This second link-sharing goal simply states that when the distribution of bandwidth is not constrained by the hierarchical link-sharing structure, that distribution should nevertheless not be arbitrary, but should follow some acceptable policy. The guidelines for the distribution of excess bandwidth should reflect both these higher-level policy concerns and realistic limitations imposed by efficiently-implemented schedulers. The distribution of excess bandwidth for the scheduler in our simulator is discussed in Appendix A.2.

Note that the link-sharing goals do not attempt to address all of 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 a class remains. For leaf classes that contain a number of aggregated connections, congestion control within the class could be provided by the use of end-to-end transport protocols such as TCP, by an explicit admissions control procedure for that class, or by a connection-based scheduling algorithm instead of FIFO scheduling within the class. For some classes of video traffic, congestion control within the class could be provided by some form of source- or receiver-based rate-adaptive congestion control. For some classes, the gateway could use RED gateway mechanisms to monitor the average queue size and provide appropriate feedback to the sources [FJ93].

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. In addition to the possibility of dynamic bandwidth allocations to existing classes, the link-sharing structure itself for a particular link can have both static and dynamic components. A static link-sharing structure with fixed classes and bandwidth allocations might be appropriate for a link shared between multiple agencies; in this case, the link-sharing bandwidth allocated to each agency might be set by the network administrator. On the other hand, a link-sharing structure with dynamic components would have provisions for the creation and removal of subclasses and for the adjustment of bandwidth allocations. Such a dynamic link-sharing structure would be appropriate when the link-sharing mechanism is used to monitor the bandwidth of specific real-time traffic flows, while at the same time ensuring that the real-time flows don't monopolize the bandwidth on the link. For dynamic link-sharing, some

mechanism is needed to limit the lifetimes of dynamically-created classes. This issue of lifetime restrictions is not addressed in this paper, but should be considered in the context of set-up protocols and admissions control procedures for real-time traffic.

3 Formal link-sharing guidelines

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

Definitions: general scheduler, link-sharing scheduler. This paper assumes that each class has its own queue at the gateway. The conceptual framework further assumes that the scheduling mechanisms include a *general scheduler* that schedules packets from leaf classes without regard to link-sharing guidelines, and a *link-sharing scheduler* that schedules packets from some leaf classes that have been exceeding their link-sharing allocations in times of congestion. We assume that all arriving packets are associated with a leaf class in the link-sharing structure. □

One job of the general scheduler is to provide for real-time traffic that has particular delay or throughput requirements. 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. We generally assume that a priority-based general scheduler is used, but this paper does not specify the general scheduler in more detail. The priorities and definitions of the classes in the class structure are a policy issue that is not addressed in this paper.

Definitions: regulated and unregulated classes. We call a class a *regulated* class if packets from that class are being scheduled by the link-sharing scheduler at the gateway; we call a class *unregulated* if traffic from the class is being scheduled by the general scheduler. Like the distinction between the general and the link-sharing scheduler, this distinction between regulated and unregulated classes is introduced for conceptual purposes, and is used by the link-sharing algorithms described later in the paper. Any implementation will have a single integrated scheduler. In general, classes will change status from regulated to unregulated, and back, as conditions in the network change. □

The link-sharing scheduler could use one of a number of algorithms to restrict the bandwidth of regulated classes. One option for the link-sharing scheduler is to rate-limit the regulated class to its link-sharing bandwidth; this regulation would be accompanied by some strategy for dropping arriving packets when necessary. However, there are other options; as an example, the link-sharing scheduler could simply decrease the priority of the regulated class, so that the general scheduler schedules packets from that class less frequently.

Definitions: classifier, estimator. In addition to the general scheduler and the link-sharing scheduler, required link-sharing mechanisms include a *classifier* and an *estimator*. The *classifier* classifies packets arriving at the gateway to the appropriate class for that output link. [WGCJF94] describes an efficient implementation of a classifier. While there are many open questions concerning the guidelines used for the classification of packets, these questions are orthogonal to scheduling issues, and will not be discussed further in this paper. In particular, we don't take a position on whether classification should be based on explicit requests for service from applications, or on packet fields (e.g., source and destination addresses, the protocol field) determined by the network.

The *estimator* estimates the bandwidth used by each class over the appropriate time interval, to determine whether or not each class has been receiving its link-sharing bandwidth. The time constant for the estimator is a critical parameter; this time constant determines the interval over which the gateway attempts to enforce the link-sharing guidelines. Appendix A discusses the estimator in more detail. □

This paper focuses on the interaction between the general scheduler and the link-sharing scheduler. In the absence of persistent congestion, the general scheduler is all that is required to schedule traffic on the output link. However, in the presence of congestion the gateway might also want to take into account link-sharing goals for sharing the link bandwidth among different traffic types, protocol families, or agencies. The link-sharing guidelines in this section specify when a class can continue unregulated and therefore scheduled by the general scheduler, and when the class should be regulated by the link-sharing scheduler.

While this paper makes a conceptual separation between the general scheduler and the link-sharing scheduler, this does not imply that the two schedulers consist of separate sections of code. For example, as Appendix A explains, in our simulator the "link-sharing scheduler" and "general scheduler" simply use different algorithms in setting a class parameter called the *time-to-send* field that indicates 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 used more than its allocated link-sharing bandwidth (in bytes/second, as averaged over a specified time interval), *underlimit* if it has used less that a *specified fraction* of its link-sharing bandwidth, and *at-limit* otherwise. The limit status of each class is determined by the estimator, and is used to determine when explicit action should be taken to correct the link-sharing behavior of the traffic. □

Note that with these definitions, if the root node of the link-sharing structure, representing the link itself, is allocated 100% of the link bandwidth, then the root class can never be overlimit.

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 exact definition should be a policy issue. The intention is that an unsatisfied class is a class that is underlimit and that has sufficient demand to use additional bandwidth. While an underlimit class that occasionally has a packet or two in the queue for a brief time should probably not be considered unsatisfied with the link-sharing behavior, in our simulator's implementation of the formal link-sharing guidelines any class with a non-empty queue is defined as having a persistent backlog. The notion of a persistent backlog is not used 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.

For a hierarchical link-sharing structure, the link-sharing guidelines are an extension of the guidelines given above. First, we define the *level* of a class in the link-sharing structure.

Definitions: levels. All leaf classes in a link-sharing structure are defined to be at *level* 1, and each interior class has a level one greater than the highest level of any of its children. An example of levels is shown in Case 1 in Figure 4. □

The formal link-sharing guidelines given below implement the link-sharing goals described in the previous section. The formal link-sharing guidelines specify when a class is allowed to *borrow* unused bandwidth from ancestor classes.

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 at level i, and there are no unsatisfied classes in the link-sharing structure at levels lower than i.
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, *unregulated*, or whether the class should have its bandwidth *regulated* by the link-sharing scheduler. The 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 contributing to the unsatisfied state of some other class in the link-sharing.

The cases below, illustrated in Figure 4, illustrate the formal link-sharing guidelines. For each example link-sharing structure in Figure 4, the bold circles mark the overlimit and underlimit classes, and small queues show which classes have a persistent backlog. Given this status, the formal link-sharing guidelines are applied to determine which classes need to be regulated. In Figure 4 the leaf classes labeled "1" are real-time classes and the classes labeled "2" are non-real-time classes.

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

Case 2: In this case there are two overlimit classes, but only the Agency B non-real-time class is unsatisfied, and the Agency A class is not overlimit. From the link-sharing guidelines, neither real-time class is allowed to borrow from their parent classes and therefore both real-time classes will be regulated. Appendix C examines the pathological behavior that could result if the Agency A real-time class was allowed to continue unregulated in this case. □

Case 3: In this case the Agency A class and the Agency A real-time class are overlimit, while the Agency B class and Agency B non-real-time class are unsatisfied. From the link-sharing guidelines, the Agency A real-time class needs to be regulated. Because the formal link-sharing guidelines take into account the limit status of ancestor classes, hierarchical link-sharing can be provided. □

Case 4: In this case no leaf classes are unsatisfied, but the Agency A class itself is unsatisfied. Because the Agency A class is underlimit and there are no unsatisfied classes at lower levels, the Agency A non-real-time class can continue unregulated. However, according to the link-sharing guidelines the Agency B real-time class should be regulated. This is another example of how the link-sharing guidelines can provide hierarchical link-sharing. □

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:

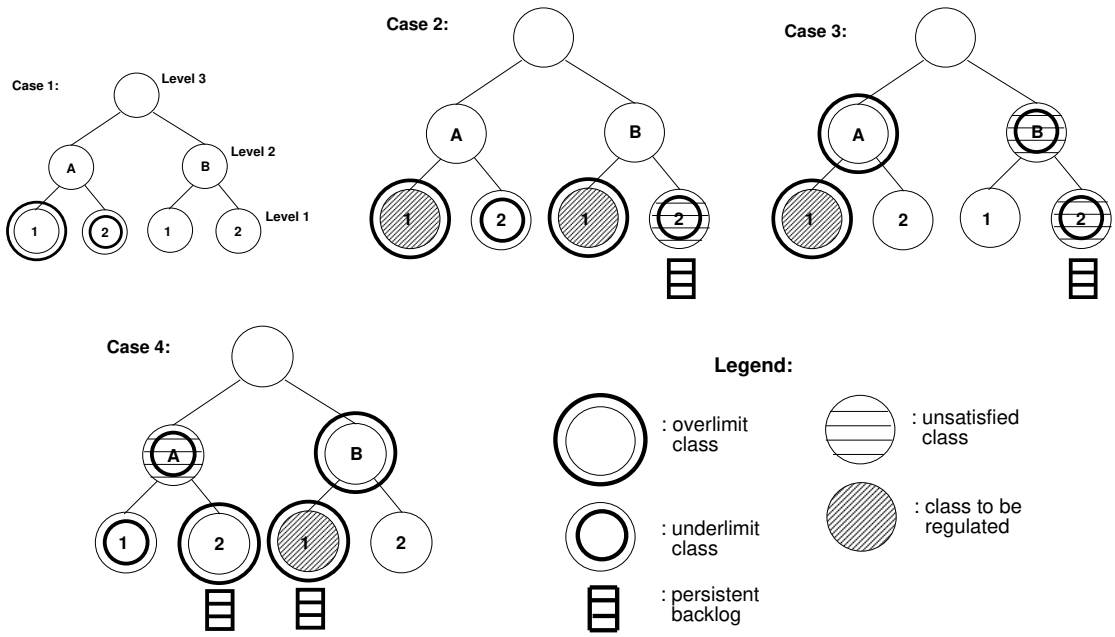

Figure 4: Examples of link-sharing scenarios.

1: The class is not overlimit, OR
2: The class has a not-overlimit ancestor at level i, and the link-sharing structure has no unsatisfied classes at levels lower than i.
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 a underlimit ancestor at level i, and the link-sharing structure has no unsatisfied classes at levels lower than i.
□

Definitions: exempt, bounded, and isolated classes. A link-sharing structure could mark some classes as either *exempt* or *bounded*, if desired. [SCZ93] introduces the notion of *exempt* traffic that is never restricted by the scheduler to its allocated link-sharing bandwidth, regardless of the level of congestion on the output link.[1] Our link-sharing structure would designate an *exempt* class by assigning the class a link-sharing bandwidth of 100% of the link bandwidth. For an *exempt* class, either the general scheduler and the admissions control procedure should ensure that the traffic from the class does not violate the link-sharing goals, or there should be a clear understanding that scheduling this traffic takes precedence over the link-sharing goals.

A *bounded* class is not allowed to borrow from ancestor classes, regardless of the limit status of those classes. This might be done, for example, for a traffic class consisting of a single high-priority real-time connection where low jitter is more important than low average delay. In our implementation of the link-sharing structure each class has both a "parent" field giving that class's parent in the class tree and a "borrow" field indicating whether or not that class is allowed to borrow unused bandwidth from the parent. A *bounded* class would have the "borrow" field set to not allow borrowing.

An *isolated* class is one that does not allow non-descendant classes to "borrow" its unused bandwidth, and that does not borrow bandwidth from other classes in turn. An *isolated* class would leave the parent field empty, and would simply be assigned a fraction of the link bandwidth. □

The formal link-sharing guidelines are discussed further in Appendix B.

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

4 Approximations to the formal link-sharing guidelines.

The previous section described a set of formal link-sharing guidelines. With the formal guidelines the decision whether or not to regulate a class depends not only on the limit status of parent classes but also on the 'satisfied' status of other classes in the same link-sharing structure. It is possible that these formal link-sharing guidelines could be efficiently implemented given appropriate architectures and/or data-structures. However, in this section we explore several approximations to formal link-sharing that lend themselves more readily to efficient implementations. This section discusses two approximations to formal link-sharing; the first approximation is Ancestors-Only link-sharing. The second approximation, Top-Level link-sharing, gives improved performance over Ancestors-Only link-sharing,

In Ancestors-Only link-sharing, for ease of implementation, the decision whether or not to regulate a class is deter-

mined only by the limit status of that class and of parent classes. A simple example with a flat link-sharing structure illustrates some of the difficulties of such an approach. For the link-sharing structure in Figure 1, 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. The most straightforward answer is to allow an overlimit class to remain unregulated only when some ancestor class is *underlimit* (instead of simply being *not overlimit*).

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 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 Agency A real-time class in Case 1 and the Agency B real-time class in Case 2. In this case, either the Agency A real-time class in Case 1 will be regulated unnecessarily, as required by the Ancestors-Only link-sharing guidelines above, or neither real-time class will be regulated and 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. Because the estimator distinguishes between at-limit and underlimit ancestor classes, allowing an overlimit leaf class to remain unregulated only when there is an underlimit ancestor, Ancestors-Only link-sharing is sensitive to the quantitative parameter used to distinguish between at-limit and underlimit classes.

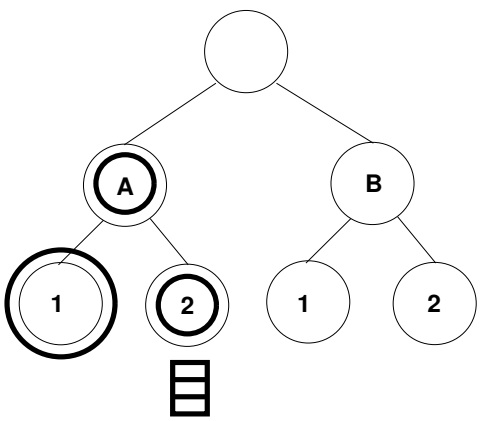

Figure 5: Sensitivity of Ancestors-Only link-sharing.

Ancestors-Only link-sharing can be sensitive to other parameters of the estimator as well. For the link-sharing structure in Figure 5, assume that, following a period when Agency B traffic used all of the link bandwidth, the priority-one class in Agency A has recently sent a large burst of packets, and has just been labeled by the estimator as overlimit. The Agency A real-time class will continue to be able to send unregulated as long as the estimator continues to label the Agency A class itself as underlimit, regardless of the unsatisfied state of Agency A's priority two class. Thus, Ancestors-Only link-sharing is sensitive to the maximum burst that can be sent before a previously-idle interior class is considered overlimit.

To illustrate some of the weaknesses of Ancestors-Only link-sharing, consider the link-sharing structure in Case 4 in Figure 4, and assume that the general scheduler gives priority to real-time traffic over non-real-time traffic. Assume further that the Agency A real-time class has little data to send for an extended period of time, and that the Agency B real-time class has unfilled demand. With Ancestors-Only link-sharing, the Agency B real-time 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, and the Agency A non-real-time class will cycle between receiving and not receiving bandwidth. 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, which use a slight modification of the Ancestors-Only approach, give a more robust approximation to formal link-sharing. In the Top-Level guidelines, the gateway still examines the limit status of ancestor classes. However, in addition the Top-Level approach considers the *levels* of the various classes in the hierarchical link-sharing structure. The gateway maintains a *Top-Level* variable that indicates the highest level from which a class is allowed to 'borrow' bandwidth. As in formal link-sharing, where classes are not allowed to borrow from ancestors at level i or above if there are unsatisfied classes at level $i-1$, Top-Level link-sharing uses the *Top-Level* variable to indicate the highest level from which a class may borrow bandwidth, and uses various heuristics to set 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. □

This is a range of possibilities for heuristics for setting the *Top-Level* variable. When *Top-Level* is set to *Infinity*, then Top-Level link-sharing is identical to Ancestors-Only link-sharing. When *Top-Level* is set to the lowest level that has an unsatisfied class, then Top-Level link-sharing is essentially the same as formal link-sharing. For example, if the gateway sets *Top-Level* to 1 when there is a class that is not overlimit and that has a nonempty queue, then for as long as *Top-Level* remains set to 1, only classes that are not overlimit will be able to send packets. However, rather

than precisely implementing formal link-sharing, and continually updating the *Top-Level* variable as queues build up and disperse and as classes change their limit status, Top-Level link-sharing avoids some of the overhead by using simplier heuristics to set the *Top-Level* variable.

Our simulator uses the guidelines below 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*.

With these guidelines, the gateway sets *Top-Level* to i only when the gateway knows that some class can send a packet without borrowing from an ancestor above level i. The setting of the *Top-Level* variable reflects partial knowledge of the gateway. For example, with these guidelines the *Top-Level* variable might be greater than 1 even though there is a not-overlimit class with a non-empty queue.

While Top-Level link-sharing requires the additional overhead in maintaining the *Top-Level* variable, compared to Ancestors-Only link-sharing, 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

5.1 Comparisons of formal, Ancestors-Only, and Top-Level link-sharing

This section illustrates the performance differences between formal, Ancestors-Only, and Top-Level link-sharing in a simulation environment with extremely simple traffic arrival patterns.

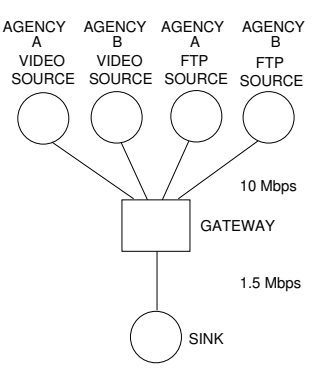

Figure 6: Simulation scenario for two-agency link-sharing.

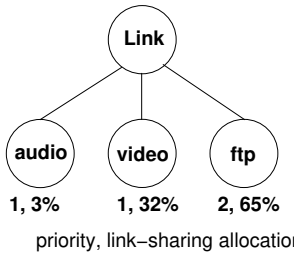

Figure 7: Link-sharing structure for the simulation of flat link-sharing.

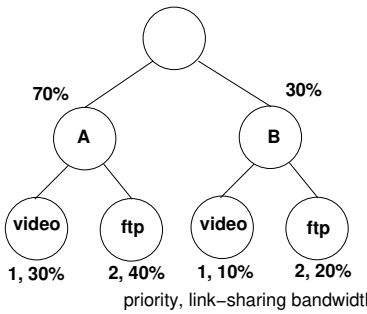

Figure 8: Link-sharing structure for the simulation of two-agency link-sharing.

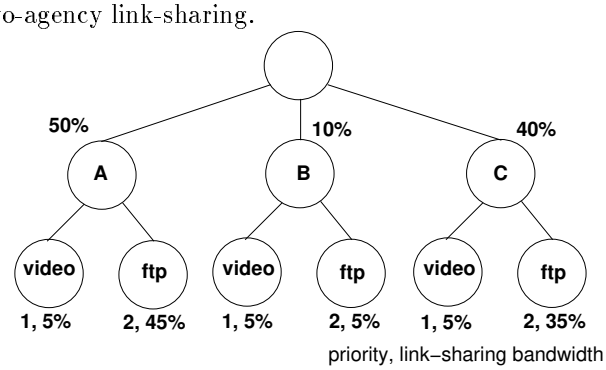

Figure 9: Link-sharing structure for the simulation of three-agency link-sharing.

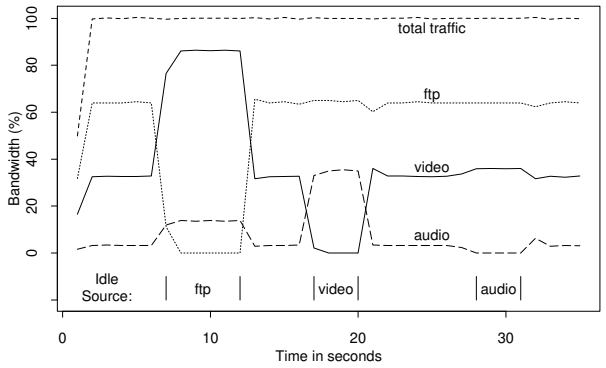

Figure 10: Flat link-sharing with Top-Level link-sharing guidelines.

Figure 6 shows the network scenario for one of the simulations. Each simulation network has a single congested gateway, with the various link-sharing structures for the

congested link shown in Figures 7 through 9. 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. In these simulations, the ftp packets are 1000 bytes. The video packets are somewhat arbitrarily set at 190 bytes, and the audio packets range from 250 to 500 bytes. The packet sizes were chosen simply to explore the behavior with a range of packet sizes for the various connections.

For the implementation of link-sharing in these simulations, the gateway maintains a separate queue for each leaf class, where each queue can hold 20 packets. Arriving packets for a class are dropped when that class's queue is full. 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 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 Appendix A. 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.[2]

For the simulation in Figure 10, the congested link uses the flat class structure in Figure 7, which has two high-priority classes and one lower-priority class. The x-axis in Figure 10 shows time and the y-axis shows the average bandwidth used by each class over one-second intervals, as a percentage of the link bandwidth. In the simulation, for each traffic class in turn the source stops transmitting for some seconds. The idle source is indicated at the bottom of the figure.

The simulation shows that when all sources are transmitting (e.g., from time 2 to time 6), each class receives roughly its link-sharing allocation. When the ftp class stops transmitting for a few seconds at time 7, the "excess" bandwidth is shared between the two real-time classes in proportion to the link-sharing bandwidths of those two classes. This sharing results from the weighted round-robin within priority levels used by the simulator's 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.

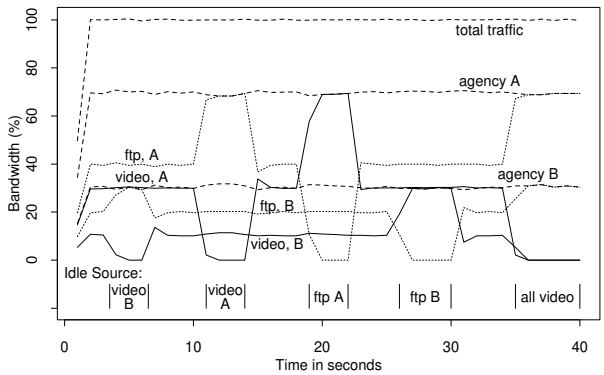

Figure 11: Two-agency link-sharing with formal link-sharing guidelines.

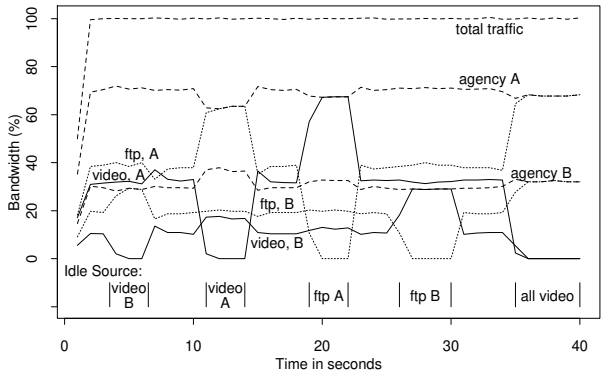

Figure 12: Two-agency link-sharing with Ancestor-Only link-sharing guidelines.

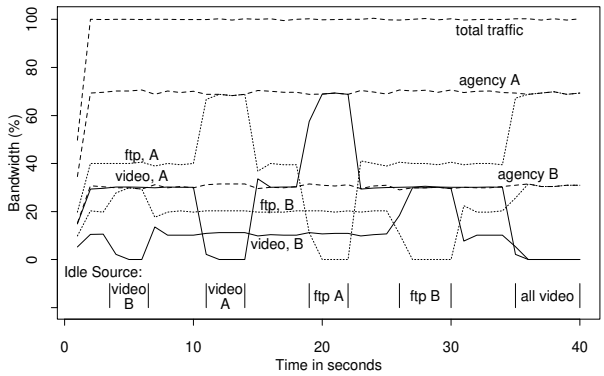

Figure 13: Two-agency link-sharing with Top-Level link-sharing guidelines.

For the simulations in Figures 11 through 13 the congested link uses the two-agency class structure in Figure 8. The figures show simulations with formal, Ancestor-Only, and Top-Level link-sharing. The solid lines show the bandwidth used by the two video classes, the dotted lines show the bandwidth used by the two ftp classes, and the dashed lines show the aggregate bandwidth used by the in-

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

terior classes. In each simulation the source for each class stops transmitting for some seconds. Note that when each 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. Near 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, when the agency A ftp class stops sending some of the "extra" bandwidth is taken by the agency B video class, and agency A receives less than its link-sharing allocation for part of the simulation. This problem is largely corrected in the simulation with Top-Level link-sharing.

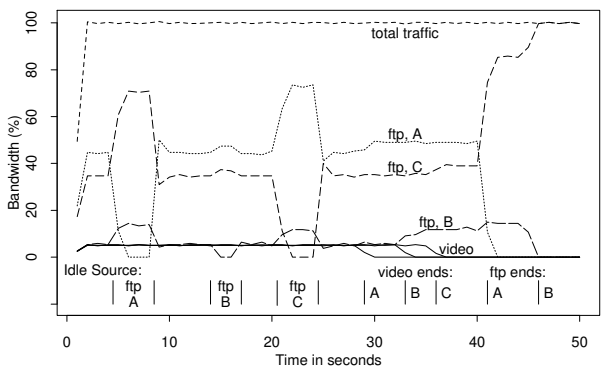

Figure 14: Three-agency link-sharing with Top-Level link-sharing guidelines.

For the simulation in Figure 14, the congested link uses the three-agency class structure in Figure 9. In this simulation all three video classes are marked as *bounded* 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.) Near the end of the simulation, one by one most of the sources stop transmitting, remaining idle for the duration of the simulation. This is shown at the bottom of the figure. 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.

5.2 Priority scheduling in a link-sharing framework

A key feature of our link-sharing framework is the ability to share bandwidth between classes with different priorities. In this section we investigate some of the interactions between priority-based schedulers and link-sharing.

The simulations in this section compare a priority-based scheduler with a non-priority-based scheduler that approximates an idealized fluid flow model of link-sharing. For these simulations, each agency has a real-time, interactive, and ftp class. The goal of the simulations is to investigate the advantages or disadvantages of giving interactive traffic priority over the ftp traffic.

Simulations of two-agency link-sharing show that when the arrival process for the interactive class is moderately bursty, the use of a priority-based scheduler that gives the interactive class priority over the ftp class can significantly reduce the delay of the interactive traffic without adversely affecting the average throughput of the ftp traffic. More precisely, the simulations show that the advantages of incorporating priorities in the link-sharing structure are most pronounced when the arrival rate for the higher-priority (interactive) class is bursty and the link bandwidth is significantly greater than the average bandwidth received by the higher-priority class.

Figure 15 shows the network scenario used for all of the simulations in this section, and Figure 16 shows the class structure for the congested link. In this paper we call a data connection *delay-sensitive* if the user is concerned with the delay of the individual packets or short bursts of packets in the connection; examples include real-time, telnet, X, and NFS traffic. We call a data connection *throughput-sensitive* if the connection transfers a fairly large number of packets, and the user is only concerned with the arrival time of the last packet in the transfer. An example of a throughput-sensitive data connection would be a file transfer where the user would like to receive the file as promptly as possible, but where the user is not concerned with the arrival time of the individual packets. For the simulations in this section, we assume that the traffic in the real-time class is constrained by an admissions control procedure, and the traffic in the interactive class is delay-sensitive but is not constrained by an admissions control procedure. We assume that the traffic in the ftp class is throughput-sensitive TCP traffic consisting of large file transfers.

For the purposes of this section, we call a class *uncongested* if there is no persistent queue, and *congested* otherwise. For each simulation in this section only one of the two ftp classes and only one of the two interactive classes is active. The active ftp class consists of three ftp connections with maximum windows set so that, in the absence of congestion, the three ftp connections together could use most of the link bandwidth. Thus, in these simulations the active ftp class is usually congested. The simulator's TCP

is based on 4.3 Tahoe TCP.

The key parameter in these simulations is the average arrival rate for the active interactive class. We are not attempting to construct a realistic source model for the interactive class; our goal is to investigate the delay and throughput of the various classes in different simulations with different arrival rates and degrees of burstiness for the interactive traffic. In the simulator the agency A interactive class has a single UDP source that generates bursts of four 1000-byte packets at exponential time intervals, while the Agency B interactive class has a UDP source that sends single 50-byte packets at exponential time intervals. (The interaction class traffic could be thought of, perhaps, as a single UDP video connection unconstrained by an admissions control procedure, or as traffic from UDP-based whiteboard sessions, or as the aggregate of a large number of short-lived connections using non-standard congestion control mechanisms.) The arrival rate for the aggregate Agency A interactive traffic is fairly bursty, while the arrival rate for the aggregate Agency B interactive traffic is fairly smooth.

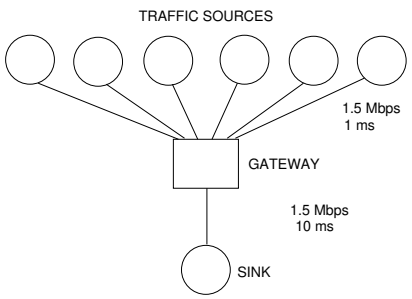

Figure 15: General simulation scenario for the investigation of delay.

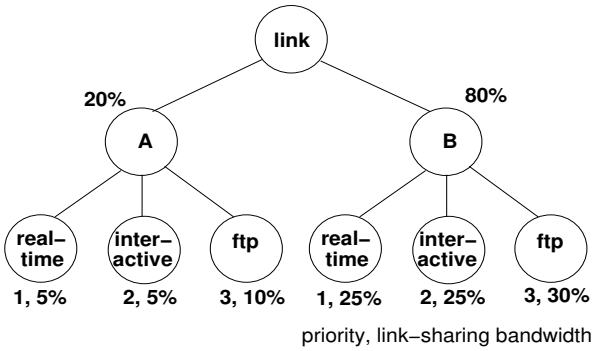

Figure 16: Link-sharing structure for the congested gateway.

Two sets of simulations were run for each scenario in this section. The first set of simulations, illustrated by a solid line in subsequent figures, uses the link-sharing structure in Figure 16. This link-sharing structure takes advantage of both bandwidth allocations and priorities for the interactive and ftp traffic. The second set of simulations, illustrated with a dashed line, uses a similar link-sharing structure, where the only change is that the ftp classes have priority two instead of priority three. This second set of simulations approximates an idealized fluid flow model of instantaneous link-sharing, such as that proposed in [SCZ93] for interactive and ftp traffic.

The simulations use the formal link-sharing guidelines. Each class has its own Drop-Tail queue at the congested gateway, where packets arriving to a full queue are dropped. The buffer size for each class is 20 packets (well over the delay-bandwidth product of four packets for a single connection). The average queueing delay for traffic in a class is determined not only by the priority level of the class, but also by the buffer size and by the fraction of bandwidth available for that class.

In the first simulation scenario, shown in Figure 17, the Agency A interactive class and the Agency B ftp class are the only active classes. Thus, the Agency A interactive class is essentially allocated 20% of the link bandwidth, and the Agency B ftp class is allocated 80%. Simulations are run for a range of values for the arrival rate of the interactive traffic, up to and exceeding the allocated rate for Agency A traffic.

In the priority-based simulation set, as long as the overall arrival rate of the interactive traffic is not large, the small bursts of interactive traffic are transmitted at the link bandwidth, rather than spread out over the fraction of the bandwidth allocated to that class.

The top graph in Figure 17 shows the average queueing delay for the interactive and ftp packets. Five forty-second simulations were run for each arrival rate for the interactive traffic, with five different seeds for the random number generator. The solid line shows the average over the five simulations for the priority-based link-sharing implementation, and the dashed line shows the average for the fluid-flow-based link-sharing implementation. The higher delay for the interactive traffic during the heaviest congestion results from the fact that while the interactive and ftp classes have buffers of the same size, the ftp class receives four times as much bandwidth as the interactive class.

The second graph in Figure 17 shows the average throughput for the interactive and ftp traffic. The dotted line shows the throughput that the interactive traffic would get if its throughput matched its arrival rate.

The third graph in Figure 17 shows the fraction of arriving packets dropped for the interactive and ftp traffic. Because of TCP's congestion control procedures, the ftp traffic can be controlled with only a small number of packet drops. For the UDP-based interactive traffic, on the other hand, as the average arrival rate exceeds the available bandwidth the fraction of packets dropped increases sharply.

The bottom graph in Figure 17 shows the ratio of the interactive packets' average queueing delay for the fluid-flow and priority-based link-sharing. For those simulations with a moderate arrival rate for the Agency A interactive traffic (less than 200 Kbps), the average delay for the interactive traffic with the fluid-flow link-sharing is four to nine times the average delay with the priority-based link-sharing, showing that for these (somewhat artificial)

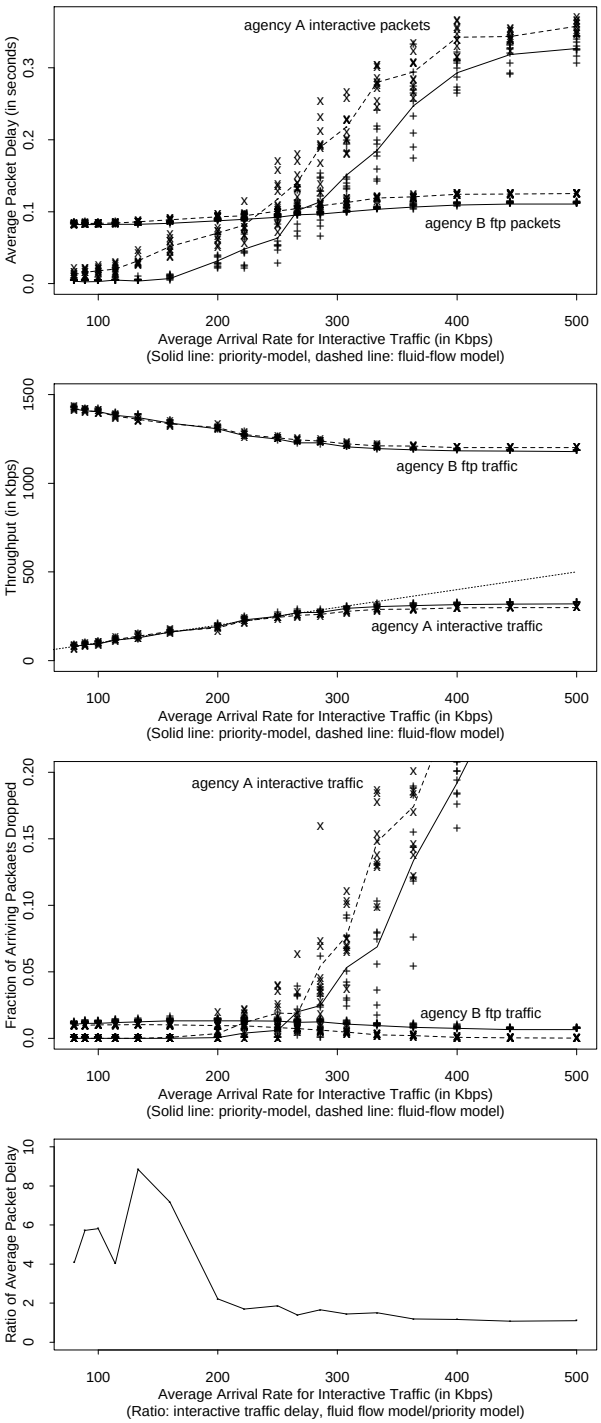

Figure 17: Scenario #1.

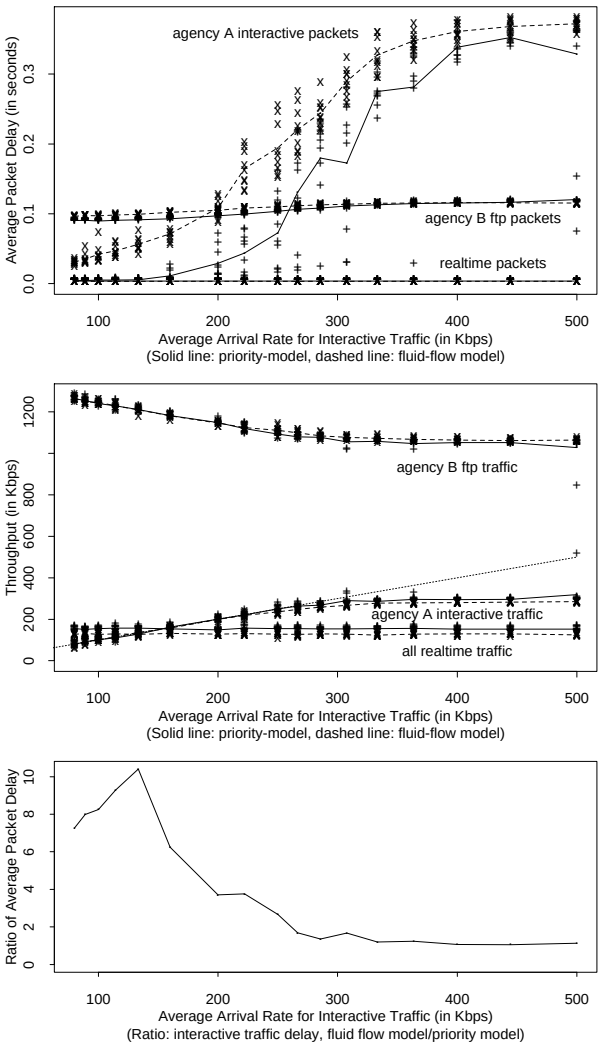

Figure 18: Scenario #2.

circumstances the advantages of the priority-based link-sharing can be significant.

In the second simulation scenario, with results shown in Figure 18, traffic is added for both real-time classes. The Agency A real-time class consists of a single ON/OFF connection, and the Agency B real-time class consists of five ON/OFF connections. Each ON/OFF connection has a peak rate of 64 Kbps (4% of the link bandwidth), and an average rate at most half of the peak rate. Thus both real-time classes are uncongested.

As Figure 18 shows, this addition of real-time traffic does not change the benefits for the interactive traffic of having priority over the ftp traffic class. The low delay of the real-time class results from its high priority and low arrival rate (lower than the allocated bandwidth).

In the third simulation scenario, with results shown in Figure 19, the Agency A ftp class and the Agency B interactive class are the only active classes. Thus, the Agency A ftp class is essentially allocated 20% of the link bandwidth, and the Agency B interactive class is allocated 80%. Given these circumstances, there is little performance difference between the priority-based and the fluid-flow-based link-sharing implementations. In either case the average delay for the delay-sensitive packets is quite low.

Thus these simulations illustrate that given a higher-priority class with bursty arrivals, coupled with a link bandwidth that is significantly larger than the average bandwidth available to the class, the use of priorities by the general scheduler can significantly limit the average delay for the higher-priority traffic, without adversely affecting

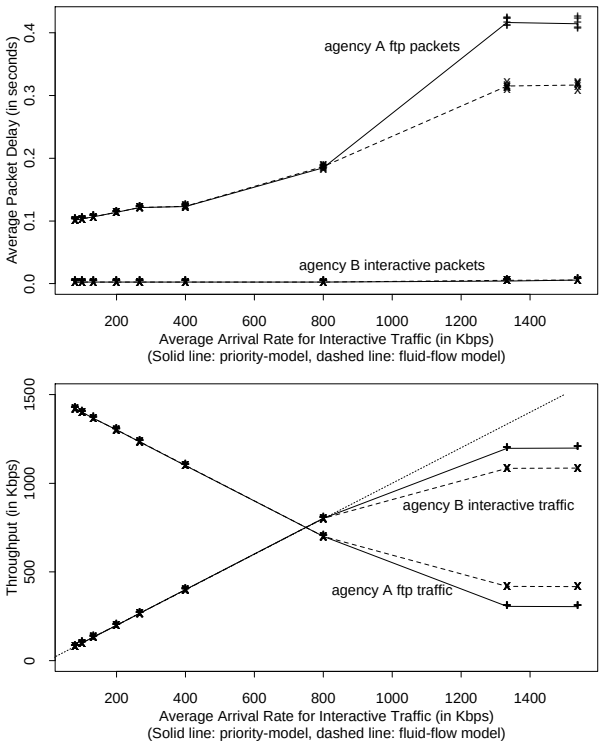

Figure 19: Scenario #3.

the average bandwidth of the lower-priority traffic. Note, however, that these simulations make no attempt to investigate whether realistic traffic scenarios are likely to fit this condition of bursty arrivals for aggregate interactive traffic.

6 Link-sharing and real-time traffic

This section considers service models for real-time traffic, paying particular attention to the relationship between link-sharing goals and real-time service models. Because link-sharing can be used to limit the bandwidth of traffic classes during times of congestion, link-sharing isolates traffic classes from each other. This isolation can be an important mechanism in accommodating emerging service models in the Internet. This isolation can also be used to protect the needs of non-real-time traffic.

We use the term *real-time traffic* to refer to traffic that has a (fixed or adaptive) playback time. As discussed below, future gateways could use explicit admissions control procedures for real-time traffic.[3] By *non-real-time traffic* we mean traffic where low delay might be desirable, but where, over reasonable time scales, the receiver waits until packets are ultimately received. We believe that one danger in current research on providing real-time services is to discount the needs of non-real-time traffic (e.g., WWW, telnet, ftp) in the Internet.

[3]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.

One possibility for a scheduler at the gateway would be to consider the needs of the real-time traffic first, and to schedule the non-real-time traffic after the needs of the real-time traffic had been met, without having the scheduler enforce bandwidth limitations on the real-time traffic [SCZ93, FBZ94]. After the needs of the real-time traffic had been met, link-sharing mechanisms would be used to share the remaining bandwidth among the non-real-time 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 accommodated by the network. Further, this sole reliance on the admissions control procedure and the policing of real-time traffic at the edge of the network is ill-suited to the needs of rate-adaptive video, and is problematic given the presence of long-range dependence in real-time traffic [GW94].

Many service models for real-time traffic assume that real-time 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, as in [FBZ94]). A slightly different service model introduced in [CSZ92] defines *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, where the receiver adapts its playback time to the delay in the network. This model of predictive service differed from previous service models in that predictive-service connections would not make an explicit agreement with the network that the connection's deterministic or statistical requirements about delay and packet loss rates will be met. Instead, the admissions control procedure for predictive service is based on measurements of past traffic in the class, and these measurements of past traffic are used as a predictor of future traffic. If the prediction of future traffic is incorrect and the predictive service class becomes oversubscribed, then this might result in increased delay and/or drop rates for the predictive service connections.

As an example of the need to protect non-real-time traffic, when the predictive service class's predictions from past traffic are incorrect there can be a conflict between the link-sharing goals and the predictive service goal of imperfectly reliable delay bounds for predictive service traffic. 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 the gateway to serve as many predictive service packets as possible, given that other real-time 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-real-time traffic over some time interval.

For a link in the core of the network with frequent changes in the number of predictive service connections, the effectiveness of the admissions control procedure is assisted by the large number of connections and the frequency with which connections come and go. The assumption that past traffic is a reliable guideline for future traffic is more effective when there are a large number of predictive service connections. In addition, for those times when the predictive service admissions control procedure is overly optimistic, in the core of the network this can be corrected simply by waiting a short time until some predictive service connections terminate.

In contrast, for a moderate-bandwidth link with a moderate number of predictive service connections the predictive service class's admissions control procedure could be less effective in protecting the link-sharing goals. In those times when the predictive service admissions control procedure has been overly optimistic, this could be corrected either by waiting until some of the (possible long-lived) predictive service connections terminate, or by having the scheduler rate-limit the predictive service traffic, dropping packets when the buffer overflows. 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 link-sharing is used to control the bandwidth of predictive service traffic during times of congestion, then this possibility should be taken into account by the admissions control procedure. In this case 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. One implication of this would be not to admit new predictive service connections when the aggregate arrival rate for the predictive service class has been exceeding the allocated predictive service bandwidth over relevant time intervals.

To illustrate the possible interaction between link-sharing and predictive service, consider a gateway with a link-sharing structure that allocates 80% of the link bandwidth to the real-time traffic and 20% to the non-real-time traffic. Assume that this is coupled with a priority-based general scheduler that gives priority to the real-time class, along with a conservative admissions control procedure for real-time class that tries to limit the real-time traffic to 50% of the link bandwidth. In this case, the link-sharing scheduler would only be used to regulate the bandwidth of the real-time class if the real-time class in fact exceeded 80% of the link bandwidth over some interval of time while the non-real-time class had unsatisfied demand. Given the goals of the admissions control procedure, this is unlikely to happen. Thus the enforcement of link-sharing at the gateway does not have to result in unacceptable service for real-time traffic. At the same time, the presence of the link-sharing mechanisms ensures that the non-real-time class will not be denied its allocated bandwidth for long intervals of time.

Like the model of predictive service, it seems likely that additional service models will emerge to meet the requirements of emerging real-time applications. One possible new application is source- or receiver-based rate-adaptive video, with a congestion control procedure to control the connection's traffic in response to congestion. In source-based rate-adaptive video, the source adjusts its transmission rate in response to feedback from the receivers, possibly using a layered coding scheme [GV93] [BTW94]. With receiver-based rate-adaptive video, the video source would partition the signal into separate layers, transmitting each layer in a separate multicast group, and receivers would unsubscribe from higher-bandwidth layers when they are experiencing congestion [M94].

These models of rate-adaptive video could be most easily accommodated 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 fraction of arriving video packets might be dropped at the gateway until the video's congestion control mechanisms respond to reduce the congestion. In a bandwidth-limited environment with video users that require a certain minimum bandwidth, a minimum-bandwidth guarantee from the network would require an admissions control procedure. For a class of video traffic without requirements of a minimum per-connection bandwidth (e.g., where the user would rather receive 1 frame each n seconds than receive no frames at all), admissions control procedures would not be required.

7 Sharing and isolation revisited

In [CSZ92] the authors propose that the addition of real-time services to packet networks be viewed in terms of the two basic principles of *isolation* and *sharing*. In this section we discuss the implications of isolation and sharing within the framework of link-sharing.

By isolating classes of traffic from each other in the network, link-sharing encourages heterogeneity, and at the same time allows connections using compatible congestion control procedures to enjoy the advantages of cooperation by sharing bandwidth in a single class of traffic. Within a leaf class, isolation between connections could be provided by a round-robin-based scheduling algorithm such as Fair Queueing [DKS90], or sharing between connections could be provided by a scheduling algorithm such as FIFO.

In [CSZ92], the authors propose that a predictive-service class should use FIFO scheduling, because FIFO scheduling reduces the tail of the delay distribution, compared to

round-robin scheduling, by reducing the delay of packets that arrive at the gateway at the end of a burst of packets from a single connection. Users within a predictive service class are protected from misbehaving users by an admissions control procedure that is coupled with policing at the edge of the network. In this section we briefly discuss some of the benefits of sharing for classes of best-effort (i.e., non-real-time) traffic, where the protection of the admissions control procedure is replaced by the protection of compatible end-to-end congestion control procedures for each connection.

One advantage of sharing and cooperation within a traffic class includes the implementation efficiencies that come from a minimum of per-connection state within the class. A second advantage is the reduction of the tail of the delay distribution that results from FIFO scheduling.

Other potential advantages of cooperation within traffic classes are less obvious, and perhaps less easy to take advantage of. For a TCP traffic class with non-cooperative scheduling algorithms such as Fair Queueing, the "fairness" of the bandwidth sharing is completely local, and doesn't take into account such factors as the number of congested gateways or the range of roundtrip times of the different connections. For a traffic class with a FIFO scheduling algorithm and cooperating connections using compatible congestion control algorithms, the bandwidth distribution between the connections can reflect such characteristics of the overall system as the range of roundtrip times and the number of congested gateways for each connection [F91]. Using an end-to-end congestion control algorithm, gateway scheduling algorithm, and gateway congestion feedback algorithm that exploit this interaction with system characteristics could perhaps improve on the purely local fairness of round-robin-based schedulers.

However, to take advantage of sharing within a link-sharing class of best-effort traffic, some form of monitoring might be appropriate to ensure that all connections in the class are in fact using acceptable congestion control procedures. As an example, with RED gateways it is straightforward to identify connections that are using a large share of the class bandwidth in times of congestion. Such classes could be isolated by being reclassified to a lower-priority or lower-bandwidth class.

8 Related work

The approach to link-sharing described in this paper is based on CBQ, an approach proposed by Van Jacobson along with other members of the End-to-end Research Group [CJ91]. For CBQ a single set of mechanisms is proposed to implement link-sharing and real-time services. The mechanisms 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. In this paper we have introduced the terms *general scheduler* and *link-sharing scheduler* as conceptual tools in exploring link-sharing algorithms. The *selector* in CBQ roughly corresponds to the *general scheduler* defined in this paper, and the *delayer* or *overlimit action* roughly corresponds to the *link-sharing scheduler* referred to in this paper.

As discussed in the Introduction, the structure of this paper was motivated in part as a response to the scheduling architecture proposed in [SCZ93]. The link-sharing scheme proposed in [SCZ93] has been discussed in Section 6. The approach in [SCZ93], after defining a service model including both quality of service commitments 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 at the gateway when commitments conflict. 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.

Our paper does not attempt to outline a complete service model, but presents an alternate approach to reconciling the quality of service commitments and the link-sharing commitments made by the gateway. 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 flow model of instantaneous link-sharing. There are two significant differences between 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 flow model does not allow for explicit link-sharing between two classes with different service priorities, such as between a real-time and a non-real-time class, or between a telnet and a lower-priority ftp class of traffic.

Second, the link-sharing in [SCZ93] takes into account the bandwidth used by real-time traffic, but the link-sharing algorithm does not affect the scheduling of the real-time 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]

In contrast, Section 6 of our paper discusses how the explicit enforcement of link-sharing goals for real-time traffic can aid the gateway in serving loss-tolerant real-time traffic (e.g., real-time traffic with drop-preference, or with rate-adaptive congestion control algorithms) by controlling the bandwidth of such traffic while providing the benefits of priority-based scheduling. Section 6 of our paper also presents our argument that this explicit enforcement of link-sharing goals for real-time traffic does not violate the goals of guaranteed or predictive real-time service, given

appropriate admissions control procedures, and contributes to the flexibility of the resource management architecture in accommodating new service models.

[SCZ93] discusses the possibility of dropping a small percentage of "preemptable" real-time packets at the gateway when other service commitments are in danger of being violated, but there is little discussion of how this could be accomplished. It is our contention that such issues should be considered within the context of hierarchical link-sharing; that is, the decision of whether or not to drop packets from a particular class should depend on whether or not that class, along with ancestor classes, is substantially contributing to congestion in the network.

Previous research on link-sharing has included simulation studies of Fair Queueing between user classes, where a user class could range from an individual application to a collection of connections associated with a particular corporation or government agency [DH90].

9 Conclusions and future work

One of the strengths of the link-sharing framework proposed in this paper is that this framework allows for priority-based or other scheduling algorithms at the gateway for real-time traffic, while incorporating provisions to protect the link-sharing behavior at the gateway. We argue that considering the link-sharing goals and real-time service goals together leads to more efficient and productive implementations of both services.

The incorporation of link-sharing mechanisms along with the provision of real-time services can simplify and add robustness to the provision of real-time services in the Internet. Link-sharing mechanisms, by isolating classes of the same priority and protecting lower-priority traffic from starvation, can add to the flexibility of the Internet in protecting best-effort traffic from higher-priority real-time traffic, or providing appropriate isolation for new congestion control protocols whose response to congestion differs from that of TCP.

There are a great number of open research questions concerning network resource management. One such open question concerns possible scenarios for the development and integration of new service models into the Internet. For example, it is not plausible that, for every potential new service model (e.g., rate-adaptive video, or perhaps reliable multicast?), a new link-sharing class will be opened for that service model on every link in the network to allow users to experiment. However, it is possible that, following somewhat the pattern of the evolution of the Mbone, some networks or links in the Internet might consider creating a new link-sharing class for a particular emerging service class, using a somewhat ad hoc classification based on fields in the packet headers. And if this turns out to be useful, then users might request other networks to create similar link-sharing classes, to expand the experience with the emerging service class, before the service class is sufficiently mature for full standardization. Certainly one of the challenges in developing resource management for the Internet will be to continue the ability to learn from working implementations deployed in a somewhat decentralized fashion (e.g., the Mbone, or the WWW). We believe that link-sharing mechanisms should be one of the components in meeting this challenge.

Acknowledgements

We thank Dave Clark, Scott Shenker, Lixia Zhang, and members of the IRTF End-to-end Research Group for many discussions (and disagreements) on link-sharing and other issues of resource management, and Jon Crowcroft, Sugih Jamin, Greg Minshall, Vern Paxson, and Joe Spagnolo for helpful feedback on various stages of this paper. Thanks also go to Steven McCanne, who has done much of the work in modifying and maintaining our simulator, and again to Sugih Jamin, who has also made contributions to our simulator.

References

[BTW94] J.-C. Bolot, T. Turletti, and I. Wakeman, "Scalable Feedback Control for Multicast Video Distribution in the Internet", *Proc. SIGCOMM '94*, August 1994, pp. 58-67.

[BCS94] B. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: an Overview", Request for Comments (RFC) 1633, IETF, June 1994.

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

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

[DH90] J. Davin and A. Heybey, "A Simulation Study of Fair Queueing and Policy Enforcement", *ACM Computer Communication Review*, Vol. 20 No. 5, pp. 23-29, Oct. 1990.

[DKS90] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm", *Internetworking: Research and Experience*, Vol. 1, 1990, pp. 3-26.

[E94] H. Eriksson, "MBone: The Multicast Backbone," *Communications of the ACM*, August 1994, Vol.37 No.8, pp.54-60.

[FBZ94] D. Ferrari, A. Banerjea, and H. Zhang, "Network Support for Multimedia: A Discussion of the Tenet Approach", to appear in Computer Networks and ISDN Systems, special issue on Multimedia Networking, 1994.

[F91] S. Floyd, Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic, *Computer Communication Re-*

view, V.21 N.5, October 1991, pp. 30-47.

[F93] S. Floyd, "Notes on Guaranteed Service in Resource Management", unpublished manuscript, March 1993.

[FJ93] S. Floyd and V. Jacobson, Random Early Detection Gateways for Congestion Avoidance, *IEEE/ACM Transactions on Networking*, V.1 N.4, August 1993, p. 397-413.

[GV93] M. Garrett and M. Vetterli, "Joint Source/ Channel Coding of Statistically Multiplexed Real-Time Services on Packet Networks", *IEEE/ACM Transactions on Networking*, V.1 N.1, 2/93, pp. 71-80.

[GW94] M. Garrett and W. Willinger, "Analysis, Modeling and Generation of Self-Similar VBR Video Traffic", *Proc. SIGCOMM '94*, August 1994, pp. 269-280.

[H94] Hoffman, D., "Implementation report on the LBL/UCL/Sun CBQ kernel", Presentation to the RSVP Working Group of the IETF, IETF, July 1994. URL http://www.ietf.cnri.reston.va.us/proceedings/94jul/tsv/rsvp.hoffman.slides.html. "An early access experimental release of Solaris RSVP/CBQ" is at URL ftp://playground.sun.com/pub/rsvp/solaris-rsvp-latest.tar.Z. Warning - This consists of 15 Mbytes of compressed code!

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

[JSZC92] S. Jamin, S. Shenker, L. Zhang, and D. Clark, "An Admission Control Algorithm for Predictive Real-time Service", *Proceedings of the Third International Workship on Networking and Operating System Support for Digital Audio and Video*, San Diego, CA, Nov. 1992, pp. 73-91.

[M94] S. McCanne, private communication, October 1994.

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

[WGCJF94] I. Wakeman, A. Ghosh, J. Crowcroft, V. Jacobson, and S. Floyd, "Implementing Real Time Packet Forwarding Policies using Streams", *Usenix 1995 Technical Conference*, January 1995, New Orleans, Louisiana, pp. 71-82. URL ftp://cs.ucl.ac.uk/darpa/usenix-cbq.ps.

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

[ZDESZ93] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, RSVP: A New Resource ReSerVaTion Protocol, *IEEE Network*, September 1993.

A Implementation in our simulator

A.1 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 efficiently implement an estimator.

The two key parameters of the estimator are the time constant for the estimator and the frequency with which the estimator updates the limit status of each class. The time constant of the estimator can be an explicit design parameter at the gateway.

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 TCP to compute the average 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.

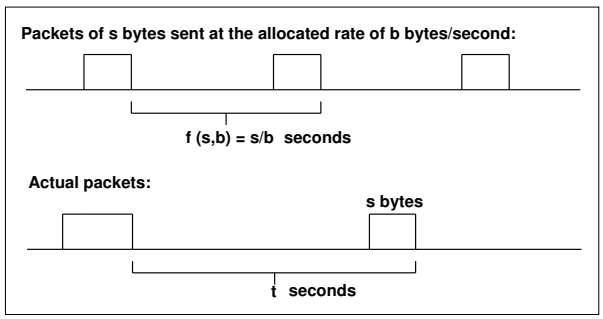

Figure 20: Variables for the computation of the limit status of a class.

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 20. 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 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 non-negative 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.

The estimator should not allow a previously-idle class to send an overly-large burst of traffic before that class is estimated as overlimit. Thus, the implementation of the estimator should explicitly consider the maximum burst that a class can send, after having been idle for a long period, before being considered overlimit. With our 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,[4] for

$$n \leq \left\lfloor \frac{\log(\frac{A}{(s/b-s/l)}+1)}{-\log(1-w)} \right\rfloor. \qquad (1)$$

Thus, 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.

The implementation of the estimator should also explicitly consider to what extent the limit status of a class should be influenced by previous bandwidth the class has received in excess of its allocated share. In our simulator, this is done with a parameter that specifies a minimum (negative) value for the variable avg.

Note that our estimator does not explicitly 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.

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. This should be adequate for $t \ll T$. 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.

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 in our simulator uses a bandwidth b' slightly less than the actual bandwidth of the link in computing the "allocated" interdeparture time $f(s,b')$.

In our simulator the value avg computed by the estimator is used to indicate the limit status of the class through the class's time-to-send field. This field indicates the next time that the gateway is allowed to send a packet from that class. For any class with avg positive, the estimator sets the time-to-send field to zero, indicating that the class is under its limit.

For a nonregulated class with avg negative (e.g., a non-leaf class), the time-to-send field is set to a time x seconds ahead of the current time, for

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

where s is the size of the packet just transmitted from the class. If the gateway waits at least x seconds before sending another packet from the class, then the class will no longer be over its limit.

For a regulated class with avg negative (e.g., a non-exempt leaf class), the link-sharing scheduler sets the time-to-send field for the class to $f(s,b)$ seconds ahead of the current time. This is the earliest time that the class will next be able to send a packet. Thus, a regulated class is

[4] 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 condition holds for values of n that satisfy Equation 1 above.

never restricted by the link-sharing scheduler to less than its allocated bandwidth, regardless of the "excess" bandwidth used by that class in the past. The link-sharing scheduler is described in Section A.3.

A.2 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, the gateway maintains a separate queue for each class associated with the 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 [F93].

The second function of the weighted round-robin is to ensure that bandwidth is distributed to unregulated classes of the same priority in proportion to the bandwidth allocations of those classes. As discussed in Section 2, the distribution of the 'excess' bandwidth among the other classes should not be arbitrary, but should follow some appropriate set of guidelines. The use of a priority-based general scheduler with weighted round-robin within priority levels results in "excess" bandwidth being distributed by the general scheduler to the higher priority classes, with the distribution proportional to the relative link-sharing allocations of those classes.

Before the general scheduler transmits a packet from a class, the scheduler checks the limit status of the 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. [WGCJF94] describes an efficient implementation of the class-based queueing mechanism that uses this general scheduler.

A.3 Implementation of the link-sharing scheduler

The link-sharing scheduler controls the scheduling of packets from regulated classes. 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 the other method might be preferred for reasons of efficiency. We describe only one of the methods in this section, involving the time-to-send field for a regulated class. The second method, more appropriate for a class that is forced to remain idle for a substantial number of packet transmission times, involves temporarily removing the class from the linked-list of classes at that priority level, and reinserting the class later after a timer expires.

For the first method in our simulator, 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, given that the packet just transmitted 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 value of avg or 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, given a priority-based general scheduler such as ours, 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 avg variable might change from negative to positive, indicating that the previously-overlimit class is no longer overlimit, and the class 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.

A.4 Publically-available CBQ distributions

[WGCJF94] contains a pointer to an unsupported version of the CBQ code, available from ftp://cs.ucl.ac.uk/darpa/cbq.tar.Z. That distribution is derived from code that predates some of the research in this paper, and essentially implements a variant of Ancestor-Only link-sharing that is not described in this paper. A subsequent unsupported version of the CBQ code is available from [H94].

B Analysis of the formal link- sharing guidelines

This appendix gives some additional discussion of the formal link-sharing guidelines given in Section 3. In particular, this section shows that as long as there is an unsatisfied leaf class, no other class will be allowed to send a packet unregulated if sending that packet would result in that class having used more that its allocated link-sharing bandwidth over the last T seconds. Section B.1 uses this to discuss limitations on starvation for lower-priority classes.

The formal link-sharing guidelines have the following properties:

- A class that is not overlimit will not be regulated. □
- When all classes are satisfied, no classes will be regulated. □

A further question concerns how long an unsatisfied class might have to wait before it begins to receive its allocated bandwidth. The bandwidth received by a particular class depends on the priority of the class, given a priority-based general scheduler, as well as on details of the estimator, general scheduler, and link-sharing scheduler. However, we can make some general observations. In this discussion, we assume that the link-sharing scheduler rate-limits regulated classes to their allocated link-sharing bandwidth.

For this section, for ease of analysis, we assume that the limit status for each class is determined by, after each packet has been transmitted, computing the bandwidth received by each class over the last T seconds.[5] Further, for simplicity, we assume that all packets are the same size, and that all bandwidth allocations translate to an integer number of packets/T-seconds. We also assume that a class is considered overlimit at time t if sending a packet at time t would result in that class having used more than its allocated link-sharing bandwidth over the last T seconds. Again, for ease of analysis, assume that a class is defined as having a "persistent backlog" whenever the queue for that class is nonempty.

We make the following assumptions about the link-sharing structure: the root class is allocated 100% of the link bandwidth, and for each non-leaf class, the sum of the bandwidth shares allocated to child classes equals the bandwidth allocated to the class itself. (That is, if a class is allocated 50% of the link bandwidth, then the sum of the allocations to the child classes also equals 50%.) We assume that no classes are marked as bounded. We further assume that the general scheduler is work-conserving; that is, the link will never be idle when there is some class with a non-empty queue.

Assume that leaf class A is unsatisfied at time t. We make the following additional observations:

- No class will be allowed to borrow from a non-leaf class for as long as class A remains unsatisfied. Thus, as long as leaf class A remains unsatisfied, the link-sharing structure is effectively the same as one where all leaf classes are children of the root class. Further, no classes will be allowed to borrow even from the root class. In this case, as long as class A remains unsatisfied, no other class will be allowed to send a packet unregulated if sending that packet results in that class using more that its allocated link-sharing bandwidth over the last T seconds. All regulated classes will be rate-limited to their allocated link-sharing bandwidth. □

B.1 Limits on starvation for lower-priority classes

In this section, in addition to assuming that leaf class A first becomes unsatisfied at time t, we assume that all other classes in the link-sharing structure have higher priority that class A. We give some quantitative bounds on how long class A could be prevented from receiving its allocated bandwidth.

It is not necessarily the case that in the first T-second interval after time t, all leaf classes will receive at most their allocated link-sharing bandwidth. For example, a previously-idle class B with higher priority, allocated a fraction f_B of the link-sharing bandwidth, could at time t send unrestricted for $f_B T$ seconds, and after that would be rate-limited to its allocated bandwidth. This class would receive more than its link-sharing bandwidth over the interval [t, $t+T$]. Note that this could occur only if the arrival rate of class B or of higher-priority classes changed significantly at time t.

The following claim explores the limits on the bandwidth that leaf classes can receive while there is an unsatisfied leaf class.

Claim 1 *Assume that leaf class A first becomes unsatisfied at time t. For every class C_i other than class A, there is a time t_i, with $t \le t_i < t + T$, such that class C_i receives at most its allocated bandwidth over every interval [t_i, t_A] during which class A remains unsatisfied. Further, if $t_i > t$, then class C_i also receives at most its allocated bandwidth over the interval [$t_i - T$, t_i], where $t_i - T < t$.*

Proof: Case 1: First, consider a class C that is overlimit at time t. Then, over every interval [t, t_A] during which class A remains unsatisfied, class C receives at most its allocated bandwidth.

Case 2: Next, consider a class C that is not overlimit at time t, and assume that class C first becomes overlimit later at time t_2. Because class C begins to be rate-limited to its

[5] We choose this method for the estimator because it seems easy to analyze, but the link-sharing behavior is more robust with an EWMA-based estimator. Unlike a plain moving-average estimator, an EWMA-based estimator can place more limitations on the bandwidth that can be used by a previously-idle class before it is declared overlimit.

allocated bandwidth at time t_2, and after time t_2 only gets to send unregulated if it is not overlimit, therefore for every interval [t_2, t_A] during which class A remains unsatisfied, class C receives at most its allocated bandwidth. Further, because class C is not overlimit at any time in the interval [t, t_2], class C received at most its allocated bandwidth over every T-second interval that ends at a time between t and t_2, inclusive.

Case 2a: Assume that class C first becomes overlimit at time $t_2 \geq t+T$. Then class C received at most its allocated bandwidth over every T-second interval in [t, t_2], and over the interval [t_2, t_A]. Therefore, over every interval [t, t_A] during which class A remains unsatisfied, class C receives at most its allocated bandwidth.

Case 2b: Assume that class C first becomes overlimit at time $t_2 < t + T$. Then class C received at most its allocated bandwidth over the T-second interval [$t_2 - T$, t_2], and over every interval [t_2, t_A] during which class A remains unsatisfied. □

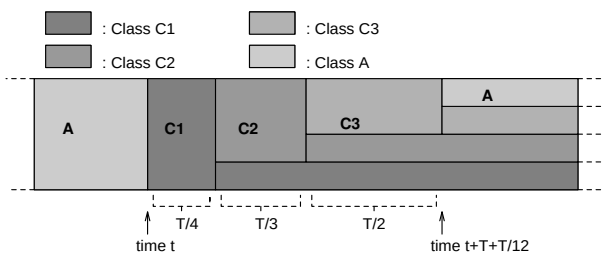

Figure 21: Class A denied bandwidth for $T+T/12$ seconds

Example: limited starvation for class A. As an example of the limits on the starvation that is possible for a lower-priority class, Figure 21 shows the bandwidth received on a link with a link-sharing structure with four classes, C_1, C_2, C_3, and A, with priorities 1, 2, 3, and 4 respectively, with each class allocated 1/4 of the link bandwidth. Class A had been receiving all of the link bandwidth, and then at time t classes C_1, C_2, and C_3 each begin to have high demand. Class $C1$ receives all of the link bandwidth for $T/4$ seconds, after which it is regulated to its link-sharing bandwidth. Similarly, classes C_2 and C_3 both get to transmit as shown in Figure 21, and class A receives no bandwidth for more than T seconds. And it is possible to construct pathological link-sharing allocations and arrival patterns where class A could wait significantly longer before receiving any bandwidth. □

Thus, depending on the scheduling algorithms, and given n classes in the link-sharing structure with link-sharing allocations $p_1, .., p_n$ (where class A is assigned link-sharing allocation p_n), it is possible for class A to receive no bandwidth at all for

$$\sum_{i=1}^{n-1} \frac{p_i}{1 - \sum_{j=1}^{i-1} p_j} T$$

seconds after time t. As an example, if each of the n classes is allocated a fraction $1/n$ of the link bandwidth, and class A has the lowest priority, then for $n = 1000$, class A could receive no bandwidth at all for $6.5T$ seconds, given arrival patterns planned by an adversary.

However, even this limited starvation of lower-priority classes is dependent on a pathological arrival process for the higher-priority classes. If the arrival process for higher-priority classes did not abruptly change at time t, then classes that are overlimit at time t would continue to be rate-limited, and classes that are underlimit at time t would continue to be underlimit. Thus, if the arrival rates for other classes did not abruptly change at time t, class A would begin to receive its link-sharing bandwidth soon after time t.

Note that the delay that a lower-priority class can receive is more limited if, as in the implementation in our simulator, the estimator explicitly limits the bandwidth that a previously-idle class can receive before being declared overlimit.

Further note that this potential for limited starvation only applies to classes that are not of the highest priority level. A class of the highest-priority is never forced to wait for higher-priority classes. The bandwidth received by the highest-priority classes is determined by the general scheduler, with the only constraint that the class might be rate-allocated to its link-sharing bandwidth in times of congestion.

C A pathological case for Ancestor-Only link-sharing

In this section we examine a pathological case that can occur in Ancestor-Only link-sharing, given an adversary who is controlling the traffic arrival pattern in order to deny bandwidth to a particular class. The example in this section also helps explain the rule in formal link-sharing that a class cannot borrow from a not-overlimit ancestor at level i if there is an unsatisfied class at level $i-1$, even if that class is not a descendant of the ancestor class.

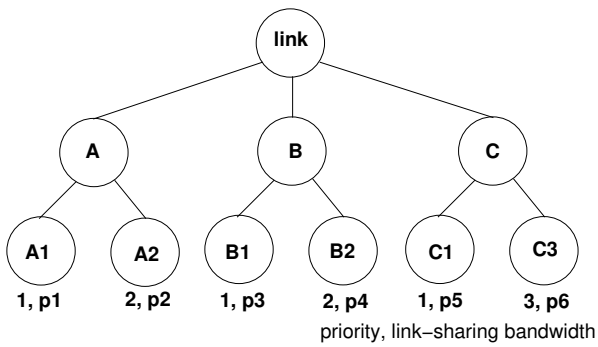

Figure 22: Example link-sharing structure.

Consider the link-sharing structure in Figure 22, and consider a priority-based general scheduler with Ancestor-Only link-sharing. Further, consider the following (highly unrealistic) packet arrival process. Assume that, at time t, packets arrive for both class A1 and class C3, and further assume, for simplicity, that the link class is at-limit, having transmitted only packets from class C1 in the last T

seconds. Class A1 is allowed to transmit for p_1T seconds because it is not overlimit, and then for p_2T additional seconds because the parent class A is underlimit. Because the root class is at-limit, class A1 is not allowed to borrow from the root class. Assume, conveniently enough, that no more packets arrive for class A1, and that class A1 now has an empty queue.

Assume that packets then arrive for class $A2$. Class $A2$ is allowed to transmit for p_2T seconds. Again assume that no more packets arrive for class $A2$, and that after p_2T seconds class $A2$ again has an empty queue.

The same process repeats for Agency B. Class B1 transmits for $(p_3 + p_4)T$ seconds, and then class B2 transmits for p_4T seconds. Then packets arrive for class C1, which by that time is underlimit, and class C1 transmits for p_5T seconds.

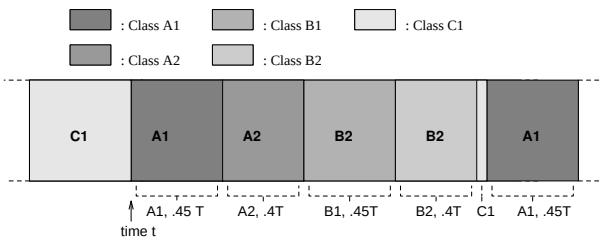

Figure 23: Class C2 denied bandwidth indefinitely, with Ancestor-Only link-sharing.

Consider the following allocations: Let p_1, p_3, p_5, and p_6 be 0.05, and let p_2 and p_4 each be 0.40. The bandwidth on the congested link is shown in Figure 23, with classes A1, A2, B1, and B2 transmitting in turn after time t. Then, after C1 has transmitted for p_5T seconds, Agency A is no longer overlimit (because no packets from Agency A have been transmitted in the last $(p_3 + 2p_4 + p_5)T = 0.9T$ seconds). If this pathological traffic arrival process repeats, then the cycle continues to repeat itself. Class A1 again sends for $(p_1+p_2)T$ seconds, at which point Agency A again becomes overlimit. Then class A2 sends for p_2T seconds, and then Agency B classes send, and so on, and class C3, being of lower priority, never sets to send any packets.

We emphasize that this pathological case requires Ancestor-Only link-sharing and a highly artificial packet arrival pattern, and is helped by a moving-average estimator without exponential decay. For example, consider what would happen if, with a more realistic packet arrival pattern, packets continued to arrive for class A1 after A1 had transmitted packets for $(p_1 + p_2)T$ seconds. Class A1 would be regulated to its link-sharing allocation of a fraction p_1 of the link bandwidth, and the cycle of class A1 receiving all of the link bandwidth, and then receiving none of the link bandwidth, would be broken. Similarly, if classes A2, B1, and B2 continued to have arriving packets after their bursts of using the link bandwidth, then those classes also would continue to be regulated to their link-sharing allocations, and there would be no starvation of class $C3$.

Note that the pathological behavior described above cannot occur with the formal link-sharing guidelines, because in that case children are not allowed to borrow from parent classes while a leaf class is unsatisfied.