A Hierarchical Fair Service Curve Algorithm for Link-Sharing, Real-Time and Priority Services

Ion Stoica Hui Zhang T. S. Eugene Ng

September 1997

CMU-CS-97-154

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

An earlier version of this paper appeared in *Proceedings of ACM SIGCOMM'97.*

This research was sponsored by Defense Advance Research Projects Agency (DARPA) under contract numbers N66001-96-C-8528 and N00174-96-K-0002, and by a National Science Foundation (NSF) Career Award under grant number NCR-9624979. Additional support was provided by Intel Corp., MCI, and Sun Microsystems.

Keywords: Resource management, scheduling, link-sharing, real-time, fairness.

Abstract

In this paper, we study hierarchical resource management models and algorithms that support both link-sharing and guaranteed real-time services with decoupled delay (priority) and bandwidth allocation. We extend the service curve based QoS model, which defines both delay and bandwidth requirements of a class, to include fairness, which is important for the integration of real-time and hierarchical link-sharing services. The resulting *Fair Service Curve link-sharing* model formalizes the goals of link-sharing and real-time services and exposes the fundamental tradeoffs between these goals. In particular, with decoupled delay and bandwidth allocation, it is impossible to simultaneously provide guaranteed real-time service and achieve perfect link-sharing. We propose a novel scheduling algorithm called Hierarchical Fair Service Curve (H-FSC) that approximates the model closely and efficiently. The algorithm always guarantees the performance for leaf classes, thus ensures real-time services, while minimizing the discrepancy between the actual services provided to the interior classes and the services defined by the Fair Service Curve link-sharing model. We have implemented the H-FSC scheduler in the NetBSD environment. By performing simulation and measurement experiments, we evaluate the link-sharing and real-time performances of H-FSC, and determine the computation overhead.

1 Introduction

The emerging integrated services networks will support applications with diverse performance objectives and traffic characteristics. While most of the previous research on integrated services networks has focused on guaranteeing QoS, especially real-time requirements, for each individual session, several recent work [1, 6, 12] has argued that it is also important to support hierarchical link-sharing service.

With hierarchical link-sharing, there is a class hierarchy associated with each link that specifies the resource allocation policy for the link. A class represents some aggregate of traffic streams that are grouped according to administrative affiliation, protocol, traffic type, or other criteria. Figure 1 shows an example class hierarchy for a 45 Mbps link that is shared by two organizations, CMU and University of Pittsburgh (U. Pitt). Below each of the two organization classes, there are classes grouped based on traffic types. Each class is associated with a bandwidth, which is the minimum amount of service this class's traffic should receive when there are enough demands.

There are several important goals for hierarchical link-sharing service. First, each class should receive certain minimum bandwidth if there are enough demands. In the example, CMU's traffic should receive at least 25 Mbps bandwidth during a period when the aggregate traffic from CMU has a higher arrival rate. Similarly, if there are resource contentions between traffic classes within CMU, the video traffic should get at least 10 Mbps. In the case when there are only audio and video streams from CMU, the audio and video traffic should receive all the bandwidth that is allocated to CMU (25 Mbps) if the demand is high enough. That is, if certain traffic classes from CMU do not have enough traffic to fully utilize its minimum guaranteed bandwidth, other traffic classes from CMU will have a higher priority to use this *excess* bandwidth than traffic from U. Pitt. While the above policy specifies that CMU audio and video traffic classes should use the excess bandwidth unused by the data traffic, there is still the issue of how the excess bandwidth is distributed between the audio and video traffic classes. A second goal of hierarchical link-sharing service is then to have a proper policy to distribute the excess bandwidth unused by a class to its sibling classes.

In addition to the two goals mentioned above, it is also important to support real-time

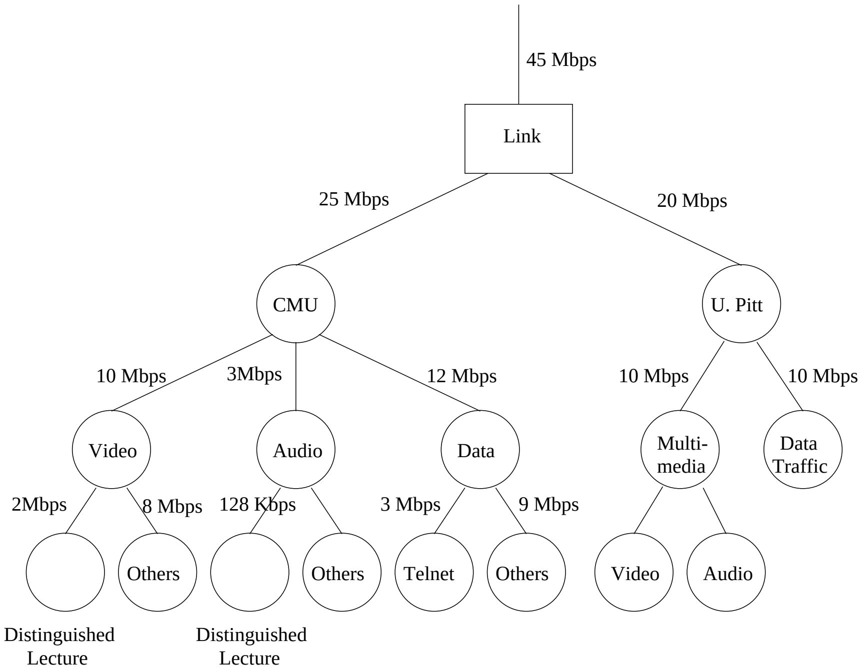


Figure 1: *An Example Link-Sharing Hierarchy.*

and priority services within the framework of hierarchical link-sharing. Since real-time service guarantees QoS on a per session basis, a natural way to integrate real-time and hierarchical link-sharing services is to have a separate leaf class for each real-time session. In the example, the CMU Distinguished Lecture video and audio classes are two leaf classes that correspond to real-time sessions. Finally, we would like to support priority service in the sense that delay (both average delay and delay bound) and bandwidth allocations are decoupled. For example, even though the CMU Distinguished Lecture video and audio classes have different bandwidth requirements, it is desirable to provide the same low delay bound for both classes. Such a decoupling of bandwidth and delay allocation is also desirable for interior or leaf classes that correspond to traffic aggregates. For example, one may want to provide a lower average delay for packets in CMU's audio traffic class than those in CMU's data traffic class.

A number of algorithms have been proposed to support hierarchical link-sharing, real-time, and priority services. However, as discussed in Section 7, they all suffer from important limitations. The fundamental problem is that with all three services, multiple requirements need to be satisfied simultaneously. This is very difficult and sometimes impossible to achieve due to conflicting requirements. This problem is exacerbated by the fact that there is no formal definition of hierarchical link-sharing service that specifies all the requirements.

In this paper, we consider an ideal model that can precisely define all the important performance goals of real-time, hierarchical link-sharing, and priority services. The basic building block of the framework is the concept of service curve, which defines a general QoS model taking into account both bandwidth and priority (delay) requirements. In this architecture, each class in the hierarchy is associated with a service curve. An ideal *Fair Service Curve link-sharing* model is to (a) simultaneously guarantee the service curves for all nodes in the hierarchy, and (b) distribute the excess bandwidth unused by a class to its sibling classes fairly. Since the service curves for class nodes are guaranteed simultaneously, the QoS for both individual sessions (leaf nodes in the hierarchy) and traffic aggregates (interior and possibly leaf nodes in the hierarchy) are satisfied. In addition, delay and bandwidth allocation can be decoupled by choosing different shapes of service curves.

Therefore, the fair service curve link-sharing model gives a precise definition of the link-sharing service that simultaneously satisfies all the important goals of real-time and link-sharing services.

Unfortunately, as will be shown in the paper, the ideal model cannot be realized at all times. In spite of this, the model serves two important purposes. First, unlike previous models, the new model explicitly defines the situations when all performance goals cannot be simultaneously satisfied, thus exposing the fundamental tradeoffs among conflicting performance goals. Second, the model serves as an ideal target that a scheduling algorithm should approximate as closely as possible.

With the ideal service model defined and the fundamental tradeoffs exposed, we propose an algorithm called Hierarchical Fair Service Curve (H-FSC) that achieves the following three goals:

- guarantee the service curves of all leaf class nodes,
- minimize the short-period discrepancy between the total amount of services provided to interior node class and its service curve,
- allocate the excess bandwidth to sibling classes, with bounded fairness

Notice that we made the architecture level decision that whenever there is a conflict, the performance guarantees of the leaf class nodes take priority. We believe this is the right tradeoff as the performance of leaf classes are most related to the performance of individual applications. In particular, since a session is always a leaf class, guaranteed real-time services can be provided on a per session basis with this framework.

The rest of the paper is organized as follows. Section 2 presents the Fair Service Curve link-sharing model and discusses the fundamental tradeoffs in approximating this model. Section 3 presents our solution, the Hierarchical Fair Service Curve (H-FSC) scheduler, followed by a discussion on its implementation complexity in Section 4. We analyze the delay and fairness properties of H-FSC in Section 5, and evaluate its performance based on both simulation and measurement experiments in Section 6. We discuss related work in Section 7 before concluding the paper in Section 8.

2 Fair Service Curve Link-Sharing Model

In this section, we first define the service curve QoS model and motivate the advantage of using non-linear service curves to decouple delay and bandwidth allocation. We then extend the concept of fairness to service curve based schedulers. Finally, we present the ideal Fair Service Curve link-sharing model and discuss the fundamental tradeoffs involved in designing a scheduler that approximates the model.

2.1 Service Curve Based QoS Model

As discussed in Section 1, we will use the service curve abstraction proposed by Cruz [4, 5] as the building block to define the idealized link-sharing model.

A session i is said to be guaranteed a service curve $S_i(\cdot)$, if for any time t_2, there exists a time $t_1 < t_2$, which is the beginning one of session i's backlogged periods (not necessarily including t_2), such that the following holds

$$S_i(t_2 - t_1) \leq w_i(t_1, t_2), \tag{1}$$

where $w_i(t_1, t_2)$ is the amount of service received by session i during the time interval $(t_1, t_2]$. For packet systems, we restrict t_2 to be packet departure times.

In the case in which the server service curve is not concave, one algorithm that supports service curve guarantees is Service Curve Earliest Deadline first (SCED) [11]. With SCED, a deadline is computed for each packet using a per session deadline curve $D_i(\cdot)$ and packets are transmitted in increasing order of their deadlines. The deadline curve $D_i(\cdot)$ is computed such that in an idealized fluid system, session i's service curve will be guaranteed if by any time t, at least $D_i(t)$ amount of service is provided to session i. Based on Eq. (1), it follows that

$$D_i(t) = \min_{t_1}(S_i(t - t_1) + w_i(t_1)), \tag{2}$$

where the minimization is over all the beginnings of session i's backlogged periods t_1's, and $w_i(t_1) = w_i(0, t_1)$ is the total amount of service session i receives till time t_1. This gives the following iterative algorithm to compute $D_i(\cdot)$. When session i becomes backlogged for the first time, $D_i(\cdot)$ is initialized to its service curve $S_i(\cdot)$. Subsequently, whenever session i becomes backlogged again at time t_a after an idling period, $D_i(\cdot)$ is updated according to the following:

$$D_i(t) = \min(D_i(t), S_i(t - t_a) + w_i(t_a)), \ \forall\, t > D_i^{-1}(w_i(t_a)). \tag{3}$$

The reason for which $D_i(\cdot)$ is defined only for $t > D_i^{-1}(c_i)$ is because this is the only portion that is used for subsequent deadline computation. Since $D_i(\cdot)$ may not be an injection, its inverse function may not be uniquely defined. Here, we define $D_i^{-1}(y)$ to be the smallest value x such that $D_i(x) = y$. Based on $D_i(\cdot)$, the deadline for a packet of length L_i^k at the head of session i's queue can be computed as follows,

$$d_i = D_i^{-1}(w_i(t) + L_i^k) \tag{4}$$

The guarantees specified by service curves are quite general. For example, the guarantees provided by Virtual Clock and various Fair Queueing algorithms can be specified by linear service curves with zero offsets.[1]. Since a linear service curve is characterized by only one parameter, the slope or the guaranteed bandwidth for the session, the delay requirement

[1] In theory, Fair Queueing and its corresponding fluid algorithm GPS can support more general service curves than linear curves [10, 15]. However, in practice, such a resource assignment has a number of limitations. See Section 7 for a detailed discussion.

cannot be specified separately. As a consequence, even though delay bounds can be provided by algorithms guaranteeing linear service curves, there is a coupling between the guaranteed delay bound and bandwidth, which results in inflexible resource allocation. With non-linear service curves, both priority (delay) and bandwidth allocation are taken into account in an *integrated* fashion, yet the allocation policies for these two resources are decoupled. This will increase the resource management flexibility and the resource utilization inside the network.

To illustrate the advantage of decoupling delay and bandwidth allocation with non-linear service curves, consider the example in Figure 2, where a video and an FTP session share a 10 Mbps link served by a SCED scheduler. Let the video source sends 30 8KB frames per second, which corresponds to a required bandwidth of 2 Mbps. The remaining 8 Mbps is reserved by a continuously backlogged FTP session. For simplicity, let all packets be of size 8 KB. Thus, it takes roughly 6.5 ms to transmit a packet. Let both video and FTP sessions be active at time 0. Then the sessions' deadline curves are also their service curves. First, consider the case in Figure 2(a) where linear service curves are used to specify the sessions' requirements. The arrival curve $A_i(\cdot)$ represents the cumulative number of bits received by session i. The deadline of a packet of session i arriving at time u is computed as the time t such that $S(t)$ equals $A(u)$. As can be seen, the deadlines of the video packets occur every 33 ms, while the deadlines of the FTP packets occur every 8.2 ms. This results in a delay of approximately 26 ms for a video packet. In the second scenario as illustrated in Figure 2(b), we use two-piece linear service curves for characterizing the sessions' requirements. The slope of the first segment of the video session's service curve is 6.6 Mbps, while the slope of the second segment is 2 Mbps. The inflection point occurs at 10 ms. The FTP session's service curve is chosen such that the entire remaining capacity is used. As can be seen, the delay of any video packet is no more than 10 ms in this case. It is important to note that the reduction in the delays for video packets does not come for free: as a result, the delays for FTP packets increase. However, this is acceptable since throughput rather than per packet delay is more important to the FTP session.

While in theory any non-decreasing functions can be used as service curves, in practice only linear or piecewise linear functions are used for reasons of simplicity. In general, a concave service curve will result in a lower average and worst case delay for a session than a linear or convex service curve with the same guaranteed assymptotic rate. However, it is impossible to have concave service curves for all sessions and still reach high average utilization. Intuitively, this is easy to understand as priority is relative and it is impossible to give all sessions high priority (low delay). Formally, the SCED algorithm can guarantee all the service curves if and only if $\sum_i S_i(t) \leq S(t)$ holds for any $t \geq 0$ where $S(t)$ is the amount of service the server provides during a time period of t. That is, the sum of the service curves over all sessions should be no more than the server's service curve.

2.2 Service Curve and Fairness

While the service curve is very general in specifying the minimum amount of service (both bandwidth and priority) guaranteed to a session or a class, it does not specify how the *excess* service, which is the extra capacity of the server beyond that is needed to guarantee the service curves of all active sessions, should be distributed. It is possible to have different scheduling algorithms that provide the same service curve guarantees but use different policies for distributing excess service. For example, while Virtual Clock and Weighted Fair Queueing (WFQ) can provide identical linear service curve guarantees, they have different fairness properties. In particular, with Virtual Clock, it is possible that a session does not receive service for an arbitrary long period because it receives excess service in a previous time period. On the contrary, the maximum period that an active session does not receive service in a WFQ server is bounded.

While the fairness property has been extensively studied for scheduling algorithms that only use sessions' rates as parameters and there are several formal definitions of fairness properties, such as the relative fairness given by Golestani [8] and the worst-case fairness given by Bennett and Zhang [2], it is unclear what fairness means and why it is important in the context of scheduling algorithms that decouple the delay and bandwidth allocation. In this section, we discuss the semantics of fairness and argue that it is important to have the fairness property even for scheduling algorithms that provides performance guarantees by decoupling the delay and bandwidth allocation. We then give a simple example to illustrate that SCED is an unfair algorithm, but can be extended to be fair.

There are two aspects of the fairness property that are of interest: (1) what is the policy of distributing excess service to each of the currently active sessions? (2) whether and to what extent a session receiving excess service in a previous time period will be penalized later?

For rate-proportional scheduling algorithms, a perfectly fair algorithm will distribute the excess service to all backlogged sessions proportional to their minimum guaranteed rates. In addition, it will not punish any session for receiving excess service in a previous time period. Generalized Processor Sharing (GPS) is such an idealized fair algorithm.

For scheduling algorithms based on general service curves, a fair algorithm should (a) distribute excess service according to a well defined policy, and (b) not penalize a session that uses excess service. Though these two aspects of the fairness property are usually considered together in a formal fairness definition, they are actually orthogonal issues. While different policies can be used to distribute excess service, in this paper we simply distribute excess service according to the service curves. It is the second aspect of the fairness property, i.e., a session that receives excess service in a previous time period should not be penalized, that we would like to emphasize in this paper.

There are two reasons why it is important to have such a fair scheduler. First, even in a network that supports guarantees, it is still desirable to let end systems to statistically share the fraction of resources that are either not reserved and/or not currently being used. A network service should encourage a source to opportunistically send more traffic than the minimum guaranteed amount, provided that the guarantees for all other sessions are not affected by the extra traffic. That is, a network should **not** penalize a session that uses more service than guaranteed if the additional service it uses is the *excess* service allotted by the server. Fairness is also important when we want to construct a hierarchical scheduler to support hierarchical link-sharing. In [1], it has been shown that the accuracy of link-sharing and delay bounds provided by Hierarchical Packet Fair Queueing (H-PFQ) is closely tied to the fairness property of PFQ server nodes used to construct the H-PFQ scheduler.

While the SCED algorithm can guarantee all the service curves simultaneously, as long as the server service curve is not concave, it does not have the fairness property. Consider the example shown in Figure 3(a). Session 1 and 2 have two-piece linear service curves $S_1(\cdot)$ and $S_2(\cdot)$, respectively, where

$$S_1(t) = \begin{cases} \alpha t, & \text{if } t \le T \\ \beta t, & \text{if } t > T \end{cases} \tag{5}$$

and

$$S_2(t) = \begin{cases} \beta t, & \text{if } t \le T \\ \alpha t, & \text{if } t > T \end{cases} \tag{6}$$

In addition, let the server rate be one, and assume the followings hold: $\alpha < \beta$, i.e., $S_1(\cdot)$ is convex and $S_2(\cdot)$ is concave, $\alpha + \beta \le 1$, i.e., both service curves can be guaranteed by using SCED, and $2\beta > 1$, i.e., it is not possible to guarantee the peak rates of both sessions simultaneously.

Also, for simplicity, assume that the packets are of unit length, and once a session becomes active it remains continuously backlogged. Under these assumptions, the deadline of the k-th packet of session i under SCED is simply $S_i^{-1}(k) + t_s^i$, where t_s^i is the time when session i becomes active. Similarly, the deadline of the last packet of session i that has been transmitted by time t ($t \ge t_s^i$) is $S_i^{-1}(w_i(t_s^i, t)) + t_s^i$. Note that since session i is not active until t_s^i, we have $w_i(t) = w_i(0, t_s^i) + w_i(t_s^i, t) = w_i(t_s^i, t)$.

Now consider the scenario in which session 1 becomes active at time 0 and session 2 becomes active at time t_0. Since session 1 is the only session active during the time interval $[0, t_0]$, it receives all the service provided by the server, i.e., $w_1(t) = t$, for any $0 \le t \le t_0$ (see Figure 3(b)). Also, the deadline of the last packet of session 1 that has been transmitted by time t_0 is $S_1^{-1}(w_1(t_0)) = S_1^{-1}(t_0)$.

Next, consider at time t_0, when the second session becomes active (see Figure 3(c)). Since the deadline of the k-th packet of session 2 is $S_2^{-1}(k) + t_0$ and packets are served in increasing order of their deadlines, it follows that as long as $S_2^{-1}(k) + t_0 < S_1^{-1}(t_0)$, only the packets of session 2 are transmitted. Thus, session 1 does *not* receive any service during the time interval $(t_0, t_1]$, where t_1 is the smallest time such that $S_2^{-1}(w_2(t_1)) + t_0 \ge S_1^{-1}(t_0)$.

As shown in Figure 3(c), for any time t, $w_1(t) > S_1(t)$ and $w_2(t) > S_2(t - t_0)$ hold, i.e., the SCED algorithm guarantees the service curves of both sessions. However, SCED punishes session 2 for receiving excess service during $[0, t_0]$ by keeping it from receiving service during $(t_0, t_1]$. This behavior makes it difficult to use SCED in a hierarchical server. To see why, consider a simple two-level hierarchy where the bandwidth is shared by two classes, characterized by the service curves $S_1(\cdot)$, and $S_2(\cdot)$, respectively. Then, if one of class 1's child classes becomes active at some point between t_0 and t_1, it will not receive any service before t_1, no matter how "important" this session is!

It is interesting to note that in a system where all the service curves are simple lines, SCED reduces to the well-known Virtual Clock discipline. While Virtual Clock is unfair [10, 16], there exists algorithms (such as the various PFQ algorithms) that not only provide the same service curve guarantees as Virtual Clock but also achieve fairness. In PFQ algorithms, each session is associated with a virtual time function that represents the normalized amount of service that has been received by the session. The algorithm then achieves fairness by minimizing the differences among the virtual time functions of all sessions. Since Virtual Clock is a special case of SCED, it is natural to use the same idea for achieving fairness in SCED with general service curves. This is achieved by associating with each session a generalized virtual time function, and servicing the session that has the smallest virtual time function. While we will describe the detailed algorithm in Section 3, we use the example in Figure 3(d) to illustrate the concept. The main modification to SCED would be to use $S_2(t - d_0)$ in computing the packets' deadlines for session 2, instead of $S_2(t - t_0)$. It can be easily verified that if $S_1(t) = r_1 t$ and $S_2(t) = r_2 t$, where r_1 and r_2 are the rates assigned to sessions 1 and 2 respectively, the above algorithm results in identical behaviors as in WFQ. Figure 3(d) shows the allocation of the service time when this discipline is used. Note that, unlike the previous case, session 1 is no longer penalized when session 2 becomes active.

In summary, fairness can be incorporated into service curve based schedulers such that (a) the excess service is distributed according to the service curves of active sessions, and (b) a session using excess service will not be penalized later. Unfortunately, this does not come for free. As shown in Figure 3(d) the service curve of session 2 is violated immediately after time t_0. This underlines the difficulty of simultaneously achieving fairness, while guaranteeing the service curves. In fact, as we will see in the next section, in general this is not possible.

2.3 Fair Service Curve Link-Sharing Model

As discussed at the beginning of the paper, the important goals of hierarchical link-sharing are: guaranteed QoS for each class, priority or decoupled delay and bandwidth allocation among classes, and proper distribution of excess bandwidth.

Since the service curve abstraction provides a general definition of QoS with decoupled delay and bandwidth allocation, and can be extended to include fairness property for the purpose of excess bandwidth distribution, it is natural to use service curves to define the performance goals of link-sharing and real-time services. In a Fair Service Curve link-sharing mode there is a service curve associated with *each* node in the link-sharing hierarchy. The goal is then to (1) satisfy the service curves of all nodes simultaneously, and (2) distribute the excess service fairly as defined in Section 2.2. Note that (1) is a general requirement that subsumes both link-sharing and real-time performance goals. A real-time session is just a leaf node in the hierarchy, and its performance will be automatically guaranteed if the Fair Service Curve link-sharing model is realized.

Unfortunately, with non-linear service curves, there are time periods when either (a) it is not possible to guarantee the service curve for all classes, or (b) it is not possible to simultaneously satisfy both the service curves and fairness property.

To see why (a) is true, consider the hierarchy in Figure 4(a). For simplicity, assume the service curve assigned to an interior class is the sum of the service curves of all its children. Also, assume all sessions are continuously backlogged from time 0 except session 1, which is idle during $[0, t]$ and becomes backlogged at time t. During $[0, t]$, since session 1 is not active, its entire service is distributed to session 2 according to the link-sharing semantics. At time t, session 1 becomes active. In order to satisfy session 1's service curve, at least $S_1(\Delta t)$ service need to be allocated for session 1 for any future time interval $(t, t + \Delta t]$. However, as shown in Figure 4(b), since the sum of all the service curves that need to be satisfied during $(t, t + \Delta t]$ is greater than the server's service curve, it is impossible to satisfy all the service curves simultaneously during this period. Since decoupling delay and bandwidth allocation is equivalent to specifying a non-linear service curve, this translates into a fundamental conflict between link-sharing and real-time service when the delay and bandwidth allocation is decoupled.

To see the fundamental conflict between fairness and real-time requirements with decoupled delay and bandwidth allocation, consider the example in Figure 3 again. As shown in Figure 3(d), if fairness is to be provided, the service curve of session 2 will be violated, i.e., $w_2(t) < S_2(t - t_0)$, for some $t \geq t_0$. This is because after t_0 both sessions receive service at a rate proportional to their slope, and since immediately after time t_0 their slopes are equal, each of them is served at a rate of $1/2$, which is smaller than β, the service rate required to satisfy $S_2(\cdot)$. Finally, it is worth to note that when all service curves degenerate to lines, this algorithm reduces to WFQ.

Therefore, there are time periods when the Fair Service Curve link-sharing model cannot be realized. In spite of this, the model serves two purposes. First, unlike previous models, this model explicitly defines the situations when all performance goals cannot be simultaneously satisfied. This exposes the fundamental architecture tradeoff decisions one has to make with respect to the relative importance among the conflicting performance goals. Second, the model serves an ideal target that a scheduling algorithm should approximate as closely as possible. We believe that a scheduler should guarantee the service curves of the *leaf* classes all the time while trying to minimize the discrepancy between the service allocated to each interior class and its fair service according to the model.

3 Hierarchical Fair Service Curve (H-FSC)

In this section, we propose a new scheduling algorithm called Hierarchical Fair Service Curve (H-FSC) that closely approximates the ideal Fair Service Curve link-sharing model as defined in the previous section.

3.1 Overview of the Algorithm

The scheduling is based on two criteria: the *real-time criteria* that ensures the service guarantee of all leaf classes, and the *link-sharing criteria* that aims to satisfy service curves of interior classes and fairly distribute the excess bandwidth. The real-time criteria is used to select the packet only if there is a potential danger that the service guarantees for leaf nodes are violated. Otherwise, the link-sharing criteria is used. Such a policy ensures the real-time guarantee of the leaf classes while at the same time minimizing the discrepancy between the actual services received by interior nodes and those defined by the ideal link-sharing model.

With H-FSC, each leaf class i maintains a triplet (e_i, d_i, v_i), while each interior class j maintains only v_j, where e_i and d_i represents the eligibility time and the deadline associated with the first packet of class i's queue, and v_i and v_j are virtual times for the classes. The deadlines are assigned such that if the deadlines of all packets of a session are met, its service curve is guaranteed. The eligibility times are used to arbitrate which one of the two scheduling criteria to use for selecting the next packet. The packet at the head of session i's queue is said to be eligible if $e_i \leq t$, where t is the current time. Eligibility times are computed such that at any given time when there are eligible packets in the system, there is a danger that the deadline of at least one packet is to be violated if the link-sharing instead real-time criteria is used, i.e., there is a potential conflict between link-sharing and real-time goals. Since the real-time goal is more important, whenever there are eligible packets, the algorithm will always use the real-time criteria, which is to select, among all eligible packets, the one with the smallest deadline. At any given time when there are no eligible packets, i.e., there are no possible conflicts between link-sharing and real-time goals, the algorithm will apply the link-sharing criteria recursively, starting from the root class and stopping at a leaf class, selects, among all child classes, the one with the smallest virtual time. While deadline and eligibility times are associated only with leaf classes, virtual times are associated with both interior and leaf classes. The virtual time of a class represents the normalized amount of service that has been received by the class. In a perfect fair system, the virtual times for all sibling classes should be identical. The objective of the link-sharing criteria is then to minimize the discrepancies between virtual times for sibling classes. The pseudo code of H-FSC is given in Figure 5. In computing eligibility time, deadline, and virtual time, the algorithm uses three curves, one for each parameter: the eligible curve $E_i(\cdot)$, the deadline curve $D_i(\cdot)$, and the virtual curve $V_i(\cdot)$. The exact algorithms to update these curves are presented Section 3.2 and Section 3.3.

There are several noteworthy points about the algorithm. First, while H-FSC needs to use two packet selection criteria to support link-sharing and real-time services, the other hierarchical algorithm, Hierarchical Packet Fair Queueing (H-PFQ) [1], selects packet solely based on the link-sharing criteria, and yet, it can support both link-sharing and real-time services. This is because H-PFQ guarantees only linear service curves, and it is feasible to guarantee all linear service curves simultaneously in a class hierarchy. In contrast, H-FSC supports decoupled delay and bandwidth allocation by guaranteeing non-linear service curves. As we have shown in Section 2, it is infeasible to guarantee all non-linear service curves simultaneously in a class hierarchy. Consequently, H-FSC uses two separate criteria for each of the link-sharing and real-time goals, and employs the mechanism of eligibility time to determine which criteria to use. Second, the algorithm uses three types of time parameters: deadlines, eligibility times, and virtual times. While leaf nodes maintain all three parameters, the interior nodes maintain only the virtual time parameter. This is because deadlines and eligibility times are used for the purpose of guaranteeing the service curves, and H-FSC provides guarantees service curves only for leaf classes. On the other hand, virtual times are used for the purpose of hierarchical link-sharing that involves the entire hierarchy, and therefore are maintained by all classes in the hierarchy. A third point to notice is that while all three parameters are time values, they are measured with respect to different clocks. Deadlines and eligibility times are *real* times in the sense that they are measured with respect to the physical real-time clock. The absolute values are important as they need to be compared with the real-time clock. In contrast, the virtual time of a class is measured with respect to the total amount of service provided by its parent class.[2] The relative differences between virtual times of sibling classes are more important than the absolute values of the virtual times. Finally, we note that in addition to the advantage of decoupling delay and bandwidth allocation by supporting non-linear service curves, H-FSC provides tighter delay bounds than H-PFQ even for class hierarchies with only linear service curves. The key observation is that in H-PFQ, packet scheduling is solely based on link-sharing criteria, which needs to go recursively from the root class to a leaf class when selecting the next packet for transmission. The net effect is that the delay bound provided to a leaf class increases with the depth of the leaf in the hierarchy [1]. In contrast, with H-FSC, the delay bound of a leaf class is determined by the real-time packet selection criteria, which considers only the leaf classes. Therefore, the delay bound is independent of the class hierarchy.

3.2 Eligible Time and Deadline

In this section, we present the algorithm to compute the deadline and the eligible time for each leaf class.

[2] For simplicity of notation, the parent of the root class is the server itself.

For each leaf class i, the algorithm maintains two curves, one for each parameter: the eligible curve $E(\cdot)$ and the deadline curve $D(\cdot)$. In addition, it keeps a variable c_i, which is incremented by the packet length each time a class i packet is selected using the real-time criteria. Thus c_i represents the total amount of service that the class has received when selected under the real-time criteria. Like SCED, the deadline curve $D_i(\cdot)$ is initialized to its service curve $S_i(\cdot)$, and updated each time session i becomes active at time t_a according to the following:

$$D_i(t) = \min(D_i(t), S_i(t - t_a) + c_i), \ \ \forall\, t > D_i^{-1}(c_i). \tag{7}$$

This is the same as Eq. (3) except that c_i is used instead of w_i. Since c_i does not change when the session receives service via the link-sharing criteria, the deadlines of future packets will not be affected due to the fact that the session receives excess service from the link-sharing hierarchy (see Figure 6). This is the essence of the "non-punishment" aspect of the fairness property.

While deadlines are used to guarantee service curves for leaf classes, eligibility times are used to arbitrate which one of the two scheduling criteria is to be applied to choose the next packet for service. The key observation is that with non-linear service curves, sometimes it is not possible to achieve perfect link-sharing and guarantee all service curves at the same time. A typical situation is when a session i with a concave service curve becomes active at t_a, joining sessions that have convex service curves. Before session i joins, the other sessions receive the excess service, but their deadline curves are not updated. When session i becomes active, if the sum of the slopes of all active sessions' deadline curves at time t is larger than the server rate, it is impossible to satisfy the service curves of all sessions.

The only solution is to have the server allocate active sessions "enough" service in advance using the real-time criteria such that the server has sufficient capacity to satisfy the service curves of all sessions when new sessions become active. However, whenever a packet is served using the real-time criteria but another packet has a smaller virtual time, there is a departure from the ideal link-sharing distribution. Therefore, to minimize the discrepancy from the ideal link-sharing model, we want to serve packets using the link-sharing criteria whenever there is no danger that the guarantees for leaf classes will be violated in the future.

In H-FSC, eligibility times are used to arbitrate which one of the two criteria is to be applied to select the next packet. To give more insight on the concept of eligibility, let $E(t)$ be the minimum service that all active sessions should receive by time t, such that irrespective of the arrival traffic, the aggregate service time required by all sessions during any future time interval $(t, t']$ *cannot* exceed $R \times (t' - t)$, i.e., cannot exceed the server capacity, R. Note that this is a necessary condition: if the active sessions do not receive at least $E(t)$ service by time t, then there exists a scenario in which the service curve of at least one session will be violated in the future. Intuitively, the worst case scenario occurs when *all* sessions are continuously active after time t [?]. Because the above condition holds for any future time t', we have

$$\begin{aligned} E(t) &= \sum_{i \in \mathcal{A}(t)} D_i(t) + [\max_{t' > t}(\sum_{i \in \mathcal{A}(t)} (D_i(t') - D_i(t)) \\ &\quad + \sum_{i \in \mathcal{P}(t)} (D_i^*(t') - D_i^*(t)) - R \times (t' - t))]^+, \end{aligned} \tag{8}$$

where D_i^* represents the deadline curve of a passive session i that becomes active at time t, and $[x]^+$ denotes $\max(x, 0)$. The above equation reads as follows. In the worst case, when all passive sessions become active at time t, the maximum service requested by all sessions during the time interval $(t, t']$ while all of them remain active is: $\sum_{i \in \mathcal{A}(t)}(D_i(t') - D_i(t)) + \sum_{i \in \mathcal{P}(t)}(D_i^*(t') - D_i^*(t))$. Since all sessions can receive at most $R \times (t' - t)$ service during the interval $(t, t']$, and since by time t the active sessions should have received at least $\sum_{i \in \mathcal{A}(t)} D_i(t)$ in order to satisfy their service curves, the above equation follows.

Thus, $E(t)$ represents the minimum service that should be allocated to the active sessions by time t using the real-time criteria in order to guarantee the service curves of all sessions in the future. The remaining (excess) service can be allocated by the link-sharing criteria. Further, it can be shown that the SCED algorithm is optimal in the sense that it can guarantee the service curves of all sessions by allocating exactly $E(t)$ service by time t. With this a possible algorithm would be simply to allocate $E(t)$ service by using SCED, and redistributing the excess service according to the link-sharing criteria. The major challenge in implementing such an algorithm is computing $E(t)$ efficiently. Unfortunately, this is difficult for several reasons. First, as shown in Eq. (8), $E(t)$ depends not only on the deadline curves of the active sessions, but also on the deadline curves of the passive ones. Since according to Eq. (7), the deadline curve depends on the time when a session becomes active, this means that we need to keep track of all these possible changes, which in the worst case is proportional to the number of sessions. Second, even if all deadline curves are two-piece linear, the resulting curve $E(t)$ can be n piece-wise linear, which is difficult to maintain and implement efficiently. Therefore, we choose to trade complexity for accuracy, by overestimating $E(t)$. The first step in the approximation is to note that (see Eq. (7)):

$$D_i^*(t') - D_i^*(t) \leq S_i(t'-t), \;\; \forall\, t' > t. \tag{9}$$

By using this inequality and the fact that $\sum_i S_i(t) \leq R \times t$, for any t, Eq. (8) becomes:

$$\begin{aligned}
E(t) &= \sum_{i \in \mathcal{A}(t)} D_i(t) + [\max_{t'>t}(\sum_{i \in \mathcal{A}(t)} (D_i(t') - D_i(t)) + \sum_{i \in \mathcal{P}(t)} (D_i^*(t') - D_i^*(t)) - R \times (t'-t))]^+ \\
&\leq \sum_{i \in \mathcal{A}(t)} D_i(t) + [\max_{t'>t}(\sum_{i \in \mathcal{A}(t)} (D_i(t') - D_i(t)) + \sum_{i \in \mathcal{P}(t)} S_i(t'-t) - R \times (t'-t))]^+ \\
&\leq \sum_{i \in \mathcal{A}(t)} D_i(t) + [\max_{t'>t}(\sum_{i \in \mathcal{A}(t)} (D_i(t') - D_i(t)) + \sum_{i \in \mathcal{P}(t)} S_i(t'-t) - \sum_{i \in \mathcal{A}(t) \cup \mathcal{P}(t)} S_i(t'-t))]^+ \\
&= \sum_{i \in \mathcal{A}(t)} D_i(t) + [\max_{t'>t}(\sum_{i \in \mathcal{A}(t)} (D_i(t') - D_i(t) - S_i(t'-t)))]^+ \\
&\leq \sum_{i \in \mathcal{A}(t)} (D_i(t) + [\max_{t'>t}(D_i(t') - D_i(t) - S_i(t'-t))]^+).
\end{aligned}$$

Finally, we define the session's eligible curve to be

$$E_i(t) = D_i(t) + [\max_{t'>t}(D_i(t') - D_i(t) - S_i(t'-t))]^+, \qquad \forall\, t > D^{-1}(c_i). \tag{10}$$

The eligible curve $E_i(\cdot)$ determines the maximum amount of service received by session i at time t by the real-time criteria. Since $\sum_{i \in \mathcal{A}(t)} E_i(t) \geq E(t)$, we have a sufficient condition. $E_i(\cdot)$ is updated every time session i becomes active by the function **update_EC** according to the above formula. It is important to note that even though the formula, which applies to algorithms with service curves of arbitrary shape, looks complicated, the eligibility curves are actually quite simple to compute in the specific cases that we are interested in. For example, for sessions with concave service curves the eligibility curve is the same as the deadline curve. Intuitively this is easy to understand as the minimum service rate for sessions with concave service curves will not increase in the future, thus there is no need to provide future service for it. Similarly, for sessions with two piece-wise linear convex service curve (first slope α, second slope β, where $\beta > \alpha$), the eligibility curve is the linear curve with the slope of β.

3.3 Virtual Time

The concept of virtual time was first proposed in the context of Packet Fair Queueing (PFQ) and Hierarchical Packet Fair Queueing (H-PFQ) algorithms to achieve fairness, real-time, and hierarchical link-sharing. In H-FSC, we will use a generalized version of virtual time to achieve hierarchical link-sharing.

Each Fair Queueing algorithm maintains a system virtual time $v^s(\cdot)$. In addition it associates to each session i a virtual start time $s_i(\cdot)$, and a virtual finish time $f_i(\cdot)$. Intuitively, $v^s(t)$ represents the normalized fair amount of service time that each session should have received by time t, $s_i(t)$ represents the normalized amount of service time that session i has received by time t, and $f_i(t)$ represents the sum between $v_i(t)$ and the normalized service that session i should receive for serving the packet at the head of its queue. Since $s_i(t)$ keeps track of the service received by session i by time t, $s_i(t)$ is also called the virtual time of session i, and alternatively denoted $v_i(t)$. The goal of all PFQ algorithms is then to minimize the discrepancies among $v_i(t)$'s and $v(t)$. In a H-PFQ system, each class keeps a virtual time function and the goal is to minimize the discrepancies among all sibling nodes in the hierarchy. Various PFQ algorithms differ in two aspects, the computation of the system virtual time function, and the packet selection policy. Examples of system virtual time functions are the start time of the packet being currently served [9], the finish time of the current packet being currently served [8], and minimum of the start times of all packets at head of currently backlogged queues [1]. Examples of packet selection policies are: Smallest Start time First (SSF) [9], Smallest Finish time First (SFF) [8], and Smallest Eligible Finish time First [1, 14]. The choice of different system virtual time functions and packet selection policies will affect the real-time and fairness properties of the resulted PFQ algorithm.

Similar to H-PFQ, for each class i in the hierarchy, H-FSC maintains a virtual time function $v_i(t)$ that represents the normalized amount of service time that class i has received by time t. In H-FSC, virtual times are used by the link-sharing criteria to distribute service along the hierarchies according to the classes' service curves. The link-sharing criteria is used to select the next packet only when the real-time criteria is not used. Since the real-time guarantees for leaf classes are ensured by the real-time packet selection criteria, the effect on performance by having different system virtual time functions and packet selection algorithms in the link-sharing criteria is less critical. In H-FSC we use the SSF policy and the following system virtual time function: $v_i^s = (v_{i,min} + v_{i,max})/2$, where $v_{i,min}$ and $v_{i,max}$ are the minimum and maximum virtual start times among all class i's currently active child classes. By doing this, we ensure that the discrepancy between the virtual times of any two active sibling sessions is bounded (see Section 5). It is interesting to note that by taking v_i^s to be either $v_{i,min}$ or $v_{i,max}$ results in a discrepancy proportional to the number of sessions.

In H-FSC, $v_i(t)$ is iteratively computed by using the previous virtual time function and the session's service curve. Virtual times are updated when a packet finishes service or a class becomes active. The function **update_v** is shown in Figure 7. Notice that **update_v** recursively updates the virtual time and the virtual time function by following child-parent link in the hierarchy till it reaches the root or a parent class that is active before time t.

In the algorithm, we actually maintain a virtual curve $V_i(\cdot)$, the inverse function of $v_i(\cdot)$, instead of $v_i(\cdot)$. $V_i(\cdot)$ is updated by using the **update_VC** function every time a class becomes active:

$$V_i(t) = \min(V_i(t), S_i(t - v_{p(i)}^s) + w_i), \;\; \forall\, t > V_i^{-1}(w_i), \tag{11}$$

where w_i is the total amount of service received by class i by time t, and $v_{p(i)}^s$ is the system virtual time for class i's parent class. Finally, it is worth noting that in the particular case when $S_i(\cdot)$ is a straight line with slope r_i, from Eq. (11) we have $V_i(t) = r_i t$. Then, the virtual time v_i is simply $V^{-1}(w_i) = w_i/r_i$, which is exactly the virtual time of session i in the PFQ algorithms.

4 Implementation Issues and Complexity

The functions **receive_packet** and **get_packet** described in Figure 5 are called each time an event occurs in the real system, i.e., a packet arrives or departs. In our current implementation we maintain two requests per session, one characterized by the eligible time and deadline, called *real-time request*, and the other characterized by the virtual time, called *link-sharing request*. For maintaining the real-time requests we can use either an augmented binary tree data structure as the one described in [13], or a calendar queue [3] for keeping track of the eligible times in conjunction with a heap for maintaining the requests' deadlines. While the former method makes possible to perform insertion and deletion (of the eligible request with the minimum deadline) in $O(\log n)$, where n is the number of active sessions, the latter method is slightly faster in the average case. The link-sharing requests are stored in a heap based on their virtual times.

Besides maintaining the request data structures, the algorithm has to compute the various curves, and update the eligible time, the deadline, and the virtual time. While it is expensive to update general service curves, in practice this complexity can be significantly reduced by considering only piece-wise linear curves.

In our model, each session i is characterized by three parameters: the largest unit of work, denoted u_i^{max}, for which the session requires delay guarantees, the guaranteed delay d_i^{max}, and the session's average rate r_i. As an example, if a session requires per packet delay guarantees, then u_i^{max} represents the maximum size of a packet. Similarly, a video or an audio session can require per frame delay guarantees, by setting u_i^{max} to the maximum size of the frame. The session's requirements are mapped to a two-piece linear service curve, which for computation efficiency is defined by the following three parameters: the slope of the first segment m_i^1, the slope of the second segment m_i^2, and the x-coordinate of the intersection between the two segments x_i. The mapping $(u_i^{max}, d_i^{max}, r_i) \rightarrow (m_i^1, x_i, m_i^2)$ for both concave and convex curves is illustrated in Figure 8.

It can be easily verified from Eq. (7) that any deadline curve that is initialized to a service curve of one of the two types discussed above remains a two-piece linear service curve after each updating operation. It is worth noting that although all two-piece linear *concave* curve exhibits this nice property, this is not true for all convex curves. In fact, it can be shown that only the two-piece linear convex service curve which have their first segment parallel with the x-coordinate have this propriety (which is our case). Since the first segment of a deadline curve does not necessarily intersect the origin, we need an extra parameter to uniquely characterize a deadline curve. We take this parameter to be the y-coordinate of the intersection between the two segments and denote it y_i. The pseudocode to update the deadline curve of session i is presented in Figure 9. The only parameters that are modified are the coordinates of the segments intersection x_i and y_i, the slopes of the two segments, m_i^1 and m_i^2, remain unchanged. It is important to note that the deadline curve, as well as the virtual and eligible curves, is updated **only** when the state of the session changes from passive to active. As long as the session remains active, no curves need to be updated.

The update operation of the virtual curve is similar to the one for the deadline curve. The only difference is that instead of using c_i and t_a, we use the total service w_i and the virtual time $v^s_{p(i)}$, respectively.

Although from Eq. (10) it appears that the computation of the eligible curve is quite complex, in our case it turns out that it can be done very efficiently: if the deadline curve is concave, then the eligible curve simply equals to the deadline curve; if the deadline curve is two-piece linear convex, then the eligible curve reduces to a line that starts from the same point with the first segment of the deadline curve, and has the same slope as its second segment.

Thus, updating the deadline, eligible and virtual curves takes constant time. Computing the eligible time, deadline and virtual time reduces to the computation of the inverse of a two-piece linear function, which takes also constant time. Consequently, H-FSC takes $O(\log n)$ per packet arrival or packet departures, which is similar to other packet scheduling algorithms [1].

5 Delay and Fairness Properties of H-FSC

In this section, we present our main theoretical results on the delay and fairness properties of H-FSC. The proofs can be found in the Appendix. For the rest of discussion, we consider the *arrival* time of a packet to be the time when its last bit was received, and the *departing* time to be the time when its last bit has been transmitted.

The following theorem shows that by computing the deadlines of each packet, based on $D_i(\cdot)$, as defined by Eq. (7), we can indeed guarantee the service curve $S_i(\cdot)$ of session i.

Theorem 1 *With H-FSC, the service curve of a session is guaranteed, if each of its packets is transmitted before its deadline.*

The next theorem gives tight delay bounds for H-FSC. In conjunction with the previous theorem, this result shows that the service curves are guaranteed within the size of a packet of maximum length.

Theorem 2 *The H-FSC algorithm guarantees that the deadline of any packet is not missed by more than* τ_{max}*, where* τ_{max} *represents the time to transmit a packet of maximum length.*

It should be noticed that, unlike H-PFQ, the delay bounds do *not* depend on the number of levels in the hierarchy. This is simply because the computation of the deadlines are based on the service curves of the leaf classes only, and packet selection using the real-time criteria is independent of the hierarchy structure.

Next, Theorem 3 characterizes the fairness of our algorithm, by giving bounds on the discrepancy in the service time distribution from the ideal link-sharing model.

Theorem 3 *In H-FSC, the difference between the virtual times of any two sibling sessions that are simultaneously active is bounded by a constant.*

From the theorem, the following corollary immediately follows:

Corollary *In H-FSC, for any two sibling classes* i *and* j *that are continuously backlogged during a time interval* $(t_1, t_2]$*, the following holds,*

$$\mid (v_i(t_2) - v_i(t_1)) - (v_j(t_2) - v_j(t_1)) \mid < B, \tag{12}$$

where B *is a positive constant.*

In other words, the difference between the normalized service time that each session should receive during the interval $(t_1, t_2]$ is bounded. It can be easily shown that when the service curves for classes i and j are linear, B reduces to the fairness metric defined by Golestani [8].

6 Performance Evaluation

We have implemented H-FSC in a simulator and in the kernel of NetBSD 1.2 on the Intel i386 architecture. We use a calendar queue in conjunction with a heap to maintain the real-time requests, and a heap at each interior class to maintain the link-sharing requests. The two implementations use nearly identical code. The only difference is that in the NetBSD implementation, we use the CPU clock cycle counter provided by the Intel Pentium Pro processor as a fine grain real-time clock for all eligible time and deadline computations. In NetBSD, besides the scheduler, we have also implemented a packet classifier that maps IPv4 packets to appropriate classes in the hierarchy.

We evaluate the H-FSC algorithm using both simulation and measurement experiments. The experiments are performed on a 200 MHz Intel Pentium Pro system with 256 KB on-chip L2 cache, 32 MB of RAM, and a 3COM Etherlink III ISA Ethernet interface card. We instrumented the kernel such that we can record a log of events (such as enqueue and dequeue) with time-stamps (using the CPU clock cycle counter) in a system memory buffer while the experiments are running, and later retrieve the contents of the log through an `ioctl` system call for post-processing and analysis. In the rest of the section, we present results to evaluate H-FSC's performance in three aspects: (1) H-FSC's ability to provide real-time guarantees, (2) H-FSC's support for link-sharing, and (3) the computation overhead of our implementation of the algorithm.

6.1 Real-time Guarantees

We use simulation to evaluate the delay properties of H-FSC because we can have better control over traffic sources in the simulator. We compare H-FSC to H-WF^{2}Q+, which, to the best of our knowledge, achieves the tightest delay bounds among all hierarchical packet fair queueing algorithms [1].

Consider the two-level class hierarchy shown in Figure 10. The value under each class represents the bandwidth guaranteed to that class. In our experiment, the audio session sends 160 byte packets every 20 ms, while the video session sends 8 KB packets every 33 ms. All the other sessions send 4 KB packets and the FTP session is continously backlogged.

To demonstrate H-FSC's ability to ensure low delay for real-time connections, we target for a 5 ms delay for the audio session, and a 10 ms delay for the video session. To achieve these objectives, we assign to the audio session the service curve $S_a = (u_a^{max} = 160 \text{ bytes}, d_a^{max} = 5 \text{ ms}, r_a = 64 \text{ Kbps})$, and to the video session the service curve $S_v = (u_v^{max} = 8 \text{ KB}, d_v^{max} = 10 \text{ ms}, r_v = 2 \text{ Mbps})$. Also, in order to pass the admission control test, we assign to the FTP session the service curve $S_{FTP} = (u_{FTP}^{max} = 4 \text{ KB}, d_{FTP}^{max} =$

16.25 ms, $r_{FTP} = 5$ Mbps). The service curves of all the other sessions and classes are linear.

Figure 11 shows the delay distribution for the audio and video sessions under H-WF^{2}Q+ and H-FSC. Clearly, H-FSC achieves much lower delays for both audio and video sessions. The reduction in delay with H-FSC is especially significant for the audio session. This is a direct consequence of H-FSC's ability to decouple delay and bandwidth allocation. The periodic variation in the delay, especially under H-WF^{2}Q+ , mirrors the periodic activity of the ON-OFF source. H-WF^{2}Q+ is more sensitive to these variations due to the coupling between bandwidth and delay allocation. Intuitively, when the ON-OFF source becomes active, the number of packets from competing sessions that an audio or video packet has to wait before receiving service almost doubles and the delay increases accordingly.[3] On the other hand, H-FSC ignores the class hierarchy in satisfying the delay requirements. Therefore, when the ON-OFF session becomes active, the number of additional packets from competing sessions an audio or video packet has to wait before being transmitted increases by less than 20 % because the bandwidth of the ON-OFF session accounts for only 18 % of the total bandwidth.

6.2 Link-sharing

To evaluate H-FSC's support for link-sharing, we conduct the following experiment using our NetBSD/i386 implementation as the platform.

We set up a class hierarchy similar to the one in Figure 10 except that there are only 4 sessions at each level. The sessions at level one all have bandwidth reservation of 1.5 Mbps, and the sessions at level two have bandwidth reservations of 80 Kbps, 480 Kbps, 1.44 Mbps and 2 Mbps respectively. The total aggregate bandwidth reservation is 10 Mbps – Ethernet's theoretical maximum throughput. All sessions are continuously backlogged except for the 2 Mbps session which is an ON-OFF source. The traffic load is generated by a self-timed user-level program that sends UDP packets of size 512 bytes for each session at the required rates. Figure 12 shows the bandwidth vs. time graph for four sessions at

[3]Because the bandwidth of the ON-OFF session accounts for 40 % of the total bandwidth of class A, when the ON-OFF session becomes active, the number of packets of class A that have deadlines during a time interval also increases by approximately 40 %.

level 2 in the hierarchy. To compute the bandwidth, a 37.5 ms averaging interval is used for all sessions except that a 60 ms interval is used for the 80 Kbps session due to its low packet rate. As can be seen, when the 2 Mbps ON-OFF session is idle, its bandwidth is fairly distributed to the other three competing sessions, while when all sessions are active, they all received their guaranteed rates.

6.3 Computation Overhead

There are generally three types of computation overhead involved in our implementation of H-FSC: packet classification, enqueue, and dequeue.

We first measure the packet classification overhead in our NetBSD/i386 implementation. To reduce the overhead of packet classification, a hashing-based algorithm is used. As a result, under light load, only the first packet of a class incurs the cost of full classification. Subsequent packets from this class are classified based on the class's hash values. While the worst-case overhead in our implementation increases with the number of classes in the hierarchy, the average time to classify a packet based on hashing is about 3 μs.

To measure the enqueue and dequeue overhead, we run the simulator in single user mode on a 200 MHz Pentium Pro system with 256 KB L2 cache and 32 MB of memory running the unchanged NetBSD 1.2 kernel. Since identical code is used in both the simulator and the NetBSD kernel implementation, the results also reflect the overhead in the NetBSD implementation.

In all experiments presented in this section, we measure (1) the average enqueue time, (2) the average dequeue time for packet selection by both the link-sharing and the real-time criteria, and (3) the average per packet queueing overhead, which is the total overhead of the algorithm divided by the number of packets forwarded. In each case, we compute the averages over the time interval between the transmission of the 10,000-th and the 20,000-th packet to remove the transient regimes from the beginning and the end of the simulation.

In the first experiment, we use one level hierarchies where the number of sessions varies from 1 to 1000 in increments of 100. The link bandwidth is divided equally among all sessions. The traffic of each session is modeled by a two state Markov process with an average rate of 0.95 of its reserved rate. As shown in Figure 13(a), enqueue and dequeue times increase little between as the number of sessions increase from 100 to 1000 sessions. This is expected as H-FSC has a logarithmic time complexity. Based on the average per packet queueing overhead, we can estimate the throughput of our implementation. For example, with 1000 sessions, since the average per packet queueing overhead is approximately 9 μs, adding the 3 μs steady-state packet classification overhead, we expect our implementation to be able to forward over 83,000 packets per second.[4]

In the second experiment, we study the impact of the number of levels in the class hierarchy on the overhead. We do this by keeping the number of sessions constant at 1000 while varying the number of levels. We consider three hierarchies: one-level, two-level with 10 internal classes, each having 100 child classes, and three-level with each internal class having 10 child classes. As shown in Figure 13(b), the enqueue and dequeue times as well as the average per packet queueing overhead increase linearly with the number of levels. Again, this is expected since each additional level adds a fixed overhead for updating the virtual times in the hierarchy which, in our case, dominates the variable overhead that is logarithmic in the number of child classes at each level.

Finally, we consider the case when all sessions are continuously backlogged. The average enqueue time in this case is very small (less than 0.3 μs) as a packet arriving at a non-empty queue is just added at the end of the queue without invoking any other processing by the algorithm. However, both types of dequeue times increase accordingly. This is because whenever a packet arrives at an empty queue or a packet is dequeued, our algorithm moves the real-time requests that have become eligible from the calendar queue into the heap. Since in this experiments all sessions are backlogged, this cost is charged to the dequeue operations only. Nevertheless, the average per packet queueing overhead changes little. For the flat hierarchy with 1000 sessions, the average per packet overhead is 8.79 μs, while for the three-level hierarchy it is 11.54 μs.

We note that all these results are obtained with relatively untuned code. We expect that the overhead can be significantly reduced with proper optimizations.

[4]This figure does not take into account route lookup and other system related overheads.

7 Related Work

Class Based Queueing [6] and Hierarchical Packet Fair Queueing [1] are two algorithms that try to support both hierarchical link-sharing and real-time services.

A CBQ server consists of a link-sharing scheduler and a general scheduler. The link-sharing scheduler decides whether to regulate a class based on link-sharing rules and mark packets of regulated classes as ineligible. The general scheduler services eligible packets using a static priority policy.

The key difference between H-FSC and CBQ is that we adopt a formal approach in designing H-FSC. By presenting a formal model that precisely defines all the important goals of link-sharing, real-time, and priority services, we expose the fundamental tradeoffs between conflicting performance goals. This enables us to design an algorithm, H-FSC, that not only provides better and stronger real-time guarantees than CBQ, but also supports more accurate link-sharing service than CBQ. In addition, H-FSC offers much stronger protection among traffic classes than CBQ when priority is supported.

For real-time services, H-FSC provides per session delay bound that is decoupled from the bandwidth requirement while CBQ provides one delay bound for all real-time sessions sharing the link. In addition, the delay bound provided by CBQ accounts only for the delay incurred by the general scheduler, but not the delay potentially incurred by the link-sharing scheduler. Since a traffic stream that is smooth at the entrance to the network may become burstier inside the network due to network load fluctuations, the link-sharing scheduler for a router inside the network may regulate the stream. With certain regulators such as those defined in [7, 17], this regulation delay does not increase the end-to-end delay bound. However, the regulating algorithm implemented by the link-sharing scheduler in CBQ is based on link-sharing rules and is quite different from the well understood regulators defined in [7, 17]. In addition, in order for the end-to-end delay bound for a session not be affected by the regulating delay, the parameters need to be consistent among all regulators for the session in the network. In CBQ, the regulation process is affected by the link-sharing structure and policy, which are independently set at each switch. Therefore, it is unclear how end-to-end delay bound will be affected by the regulation of link-sharing schedulers.

For link-sharing service, by approximating the ideal and well-defined Fair Service Curve link-sharing model, H-FSC can identify precisely and efficiently during run-time the instances when there are conflicts between requirements of the leaf classes (real-time) and interior node classes (link-sharing). Therefore, H-FSC can closely approximate the ideal link-sharing service without negatively affecting the performance of real-time sessions. With CBQ, there could be situations where the performance of real-time sessions is affected under the Formal-Link-Sharing or even the more restricting Ancestor-Only rules [6]. To avoid the effect on real-time sessions, a more restrictive Top-Level link-sharing policy is defined.

Another difference between H-FSC and CBQ is that with H-FSC, priorities for packets are dynamically assigned based on its service curves, while with CBQ, they are statically assigned based on priority classes. In CBQ, the link-sharing rule is affected only by bandwidth; once packets become eligible, they will have a static priority. This has some undesirable consequences. As an example, consider the class hierarchy in Figure 1, assume that CMU has many active video streams (priority 1) but no data traffic (priority 2), according to the link-sharing rule, CMU video traffic will become eligible at a rate of 25 Mbps. Once become eligible, they will all be served at the highest priority by the general scheduler. This will negatively affect not only the delay bound provided to U. Pitt's real-time traffic, but also the average delay of U. Pitt's data traffic, which is served by the general scheduler at a lower priority. In contrast, H-FSC provides much stronger firewall protection between different classes. The service curve for a leaf class will be guaranteed *regardless* of the behavior of other classes. In addition, link-sharing among classes is also dictated by service curves. The excess service received by a class will be limited by its ancestors' service curves, which specifies both bandwidth and priority in an integrated fashion.

Like H-FSC, H-PFQ is also rooted in a formal framework. The major difference between H-PFQ and H-FSC is that H-FSC decouples the delay and bandwidth allocation, thus achieves more flexible resource management and higher resource utilization. In addition, unlike H-PFQ where a session's delay bound increases as the depth of the hierarchy, the delay bound provided by H-FSC is not affected by the depth of the hierarchy.

In this paper, we use service-curve based schedulers to achieve decoupling of delay and bandwidth allocation. In [10, 15], it has been shown that more general service curves than linear curves can be supported by GPS. However, this general resource assignment of GPS is only possible if *all* relevant sessions in the *entire* network be policed at the source.

Therefore, sources will not be able to opportunistically utilize the excess bandwidth by sending more traffic than reserved. It is unclear whether link-sharing can be supported in such a network. With H-FSC, the scheduler guarantees a minimum service curve to a session regardless of the behaviors of other sessions in the network. In addition, it does not require that the session's input traffic to be policed at the network entrance, thus allows sources to statistically share the excess bandwidth inside the network. Furthermore, even for real-time services that do not allow link-sharing, service-curve based schedulers will achieve a larger schedulability region than GPS with general resource assignments.

8 Conclusion

We make two important contributions. First we define an ideal Fair Service Curve link-sharing model that supports (a) guaranteed QoS for all sessions and classes in a link-sharing hierarchy; (b) fair distribution of excess bandwidth; and (c) priority service or decoupled delay and bandwidth allocation. By defining precisely the ideal service to be supported, we expose the fundamental architecture level tradeoffs that apply to *any* schedulers designed to support link-sharing, real-time, and priority services. As a second contribution, we propose a novel scheduler called H-FSC that can accurately and efficiently approximate the ideal Fair Service Curve link-sharing model. The algorithm always guarantees the performance for leaf classes while minimizing the discrepancy between the actual services provided to the interior classes and the services defined by the ideal model. We have implemented the H-FSC scheduler in the NetBSD environment, and demonstrated the effectiveness of our algorithm by simulation and measurement experiments.

References

[1] J.C.R. Bennett and H. Zhang. Hierarchical packet fair queueing algorithms. In *Proceedings of the ACM-SIGCOMM 96*, pages 143–156, Palo Alto, CA, August 1996.

[2] J.C.R. Bennett and H. Zhang. WF^2Q: Worst-case fair weighted fair queueing. In *Proceedings of IEEE INFOCOM'96*, pages 120–128, San Francisco, CA, March 1996.

[3] R. Brown. Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem. *Communications of the ACM*, 31(10):1220–1227, October 1988.

[4] R. Cruz. Service burstiness and dynamic burstiness measures: A framework. *Journal of High Speed Networks*, 1(2):105–127, 1992.

[5] R. Cruz. Quality of service guaranteed in virtual circuit switched network. *IEEE Journal on Selected Areas in Communications*, 13(6):1048–1056, August 1995.

[6] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4), August 1995.

[7] L. Georgiadis, R. Guérin, and V. Peris. Efficient network QoS provisioning based on per node traffic shaping. In *IEEE INFOCOM'96*, San Francisco, CA, March 1996.

[8] S. Golestani. A self-clocked fair queueing scheme for broadband applications. In *Proceedings of IEEE INFOCOM'94*, pages 636–646, Toronto, CA, June 1994.

[9] P. Goyal, H.M. Vin, and H. Chen. Start-time Fair Queuing: A scheduling algorithm for integrated services. In *Proceedings of the ACM-SIGCOMM 96*, pages 157–168, Palo Alto, CA, August 1996.

[10] A. Parekh. *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks*. PhD dissertation, Massachusetts Institute of Technology, February 1992.

[11] H. Sariowan, R.L. Cruz, and G.C. Polyzos. Scheduling for quality of service guarantees via service curves. In *Proceedings of the International Conference on Computer Communications and Networks (ICCCN) 1995*, pages 512–520, September 1995.

[12] S. Shenker, D. Clark, and L. Zhang. A scheduling service model and a scheduling architecture for an integrated services network, 1993. preprint.

[13] I. Stoica and H. Abdel-Wahab. Earliest eligible virtual deadline first: A flexible and accurate mechanism for proportional share resource allocation. Technical Report TR-95-22, Old Dominion University, November 1995.

[14] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. Baruah, J. Gehrke, and G. Plaxton. A proportional share resource allocation for real-time, time-shared systems. In *Proceedings of the IEEE RTSS 96*, pages 288 – 289, December 1996.

[15] Z. Liu Z.-L. Zhang and D. Towsley. Closed-form deterministic performance bounds for the generalized processor sharing scheduling discipline, 1997. To appear journal of Combinatorial Optimaization.

[16] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proceedings of the IEEE*, 83(10):1374–1399, October 1995.

[17] H. Zhang and D. Ferrari. Rate-controlled service disciplines. *Journal of High Speed Networks*, 3(4):389–412, 1994.

Appendix

In this section we prove the main theoretical results of the paper, concerning both real-time guarantees and fairness. Theorem 2 shows that the service curve of any leaf class is never violated by more than τ_{max}, where τ_{max} represents the transmission time of the packet of maximum length. This result is optimal in a system in which the packet transmission is assumed to be non-preemptable. Theorem 3 shows that the difference between the virtual times of any two sibling leaf classes is always bounded. Since the virtual time keeps track of the work progress of each class, this result shows that the discrepancy between the service actually received by a class from the service it should receive in an idealized fair system is bounded.

In the remaining of this section we assume that, unless otherwise specified, all the functions are well defined over the specified intervals. Also, we assume that any non-decreasing function $f(\cdot)$ has an "inverse", $f^{-1}(\cdot)$, defined as follows: $f^{-1}(y)$ is the *smallest* value x such that $f(x) = y$ (here, we implicitly assume that there exists such an x). Finally, we make the observation that the "inverse" of any non-decreasing function is also non-decreasing.

We start with some simple definitions and results that hold for any non-decreasing function $f(\cdot)$. Consequently, these results can be applied to any service curve, and in addition, as we will show in Lemmas 3 and 4, to the deadline and eligible curves as well.

Definition 1 *The* **envelope** *of an arbitrary function $f(\cdot)$, denoted, $f_{max}(\cdot)$ is defined as*

$$f_{max}(t) = \max_{t'}(f(t' + t) - f(t')). \tag{13}$$

In words, the value of a function envelope at t, $f_{max}(t)$, represents the maximum difference between the values of $f(\cdot)$ among all pairs of points situated at the distance t apart from each other. From the above definition we have

Lemma 1 *Consider an arbitrary function $f(\cdot)$. Then, for any time t and for any $\Delta t \geq 0$, we have*

$$f(t + \Delta t) \leq f(t) + f_{max}(\Delta t). \tag{14}$$

For convenience, in the followings we limit our discussion to non-decreasing functions only.

Definition 2 *Let $f(\cdot)$ be a non-decreasing function. Then, the* **burstiness** *associated to $f(\cdot)$, denoted $B^f(\cdot)$, is defined as*

$$B^f(t) = \max_{t'}(f(t'+t) - f(t')) - \min_{t'}(f(t'+t) - f(t')). \tag{15}$$

Also we define **maximum burstiness** *of function $f(\cdot)$, denoted B^f_{max}, as*

$$B^f_{max} = \max_{t} B^f(t). \tag{16}$$

Without loss of generality, in this paper we restrict our attention to service curves with bounded maximum burstiness. To get the intuition behind the above definitions, in Figure 14 we present the envelope and the burstiness functions associated to a two-piece linear concave function, and a two-piece linear convex function, respectively.

The following result bounds the difference between the values of a non-decreasing function $f(\cdot)$ in two arbitrary points at the distance Δt from each other, in which the function is defined.

Lemma 2 *For any non-decreasing function $f(\cdot)$ that is zero in origin (i.e., $f(0) = 0$), and any time interval $[t, \Delta t)$, we have*

$$f(\Delta t) - B^f(\Delta t) \leq f(t + \Delta t) - f(t) \leq f(\Delta t) + B^f(\Delta t). \tag{17}$$

Proof. Both the right-hand side and the left-hand side inequalities follow directly from Definition 2. More precisely, for the right-hand side, we have

$$\begin{aligned} B^f(\Delta t) &= \max_{t'}(f(t'+\Delta t) - f(t')) - \min_{t'}(f(t'+\Delta t) - f(t')) \\ &\geq f(t+\Delta t) - f(t) - \min_{t'}(f(t'+\Delta t) - f(t')) \\ &\geq f(t+\Delta t) - f(t) - (f(\Delta t) - f(0)) \\ &= f(t+\Delta t) - f(t) - f(\Delta t). \end{aligned} \tag{18}$$

Similarly, for the left-hand side inequality we have

$$\begin{aligned} B^f(\Delta t) &= \max_{t'}(f(t'+\Delta t) - f(t')) - \min_{t'}(f(t'+\Delta t) - f(t')) \\ &\geq f(\Delta t) - f(0) - \min_{t'}(f(t'+\Delta t) - f(t')) \\ &\geq f(\Delta t) - (f(t+\Delta t) - f(t)), \end{aligned} \tag{19}$$

which concludes the proof. □

Without loss of generality, in the remaining of this section, we assume that all service curves are non-decreasing functions, which are zero in origin. The next lemma shows that the deadline curve is a non-decreasing function, as well. Next, since when a session becomes active for the first time its deadline curve is initialized to the service curve, it follows that the deadline curve is also zero in origin. Finally, since the virtual curve definition is identical to the one for the deadline curve (only the points in which it is updated are different), it follows that Lemmas 1 and 2 can be applied to all service, deadline, and virtual curves.

Lemma 3 *Consider a session i characterized by the service curve $S_i(\cdot)$. Then the deadline curve $D_i(\cdot)$ is non-decreasing.*

Proof. The proof is by induction on the moments of time when session i becomes active (according to Eq. (7) these are the only times when $D_i(\cdot)$ may change).

Basic Step. First time when session i becomes active, $D_i(\cdot)$ is initialized to $S_i(\cdot)$ and therefore the lemma is trivially true.

Induction Step. Let t_a be a time when session i becomes active. For clarity let $D_i^{old}(\cdot)$ be the deadline curve of session i at t_a^-, and let $D_i^{new}(\cdot)$ be the updated deadline curve immediately after session i becomes active at time t_a. Then, according to Eq. (7), we have

$$D_i^{new}(t) = \min(D_i^{old}(t), S_i(t-t_a) + c_i), \;\; t \geq (D_i^{old})^{-1}(c_i). \tag{20}$$

Next, chose two arbitrary points t_2 and t_1, such that $t_2 \geq t_1 \geq (D_i^{old})^{-1}(c_i)$. Then, to show that the new computed deadline curve, $D_i^{new}(\cdot)$, is non-decreasing it suffices to show that $D^{new}(t_1) \leq D^{new}(t_2)$. From Eq. (20), we obtain

$$D_i^{new}(t_2) - D_i^{new}(t_1) = \min(D_i^{old}(t_2), S_i(t_2-t_a)+c_i) - \min(D_i^{old}(t_1), S_i(t_1-t_a)+c_i). \tag{21}$$

Since both $D_i^{old}(\cdot)$ and $S_i(\cdot)$ are assumed to be non-decreasing, we have $D_i^{old}(t_2) \geq D_i^{old}(t_1)$, and $S_i(t_2 - t_a) \geq S_i(t_1 - t_a)$, which gives us $D_i^{new}(t_2) \geq D_i^{new}(t_1)$. $\square$

Lemma 4 *Consider a session i characterized by the service curve $S_i(\cdot)$. Then the eligible curve $E_i(\cdot)$ is non-decreasing.*

Proof. We need to show that $E_i(t_1) \leq E_i(t_2)$, for any $t_1 < t_2$. From Eq. (10), we have

$$\begin{aligned} E_i(t_2) - E_i(t_1) &= D_i(t_2) + [\max_{t'>t_2}(D_i(t') - D_i(t_2) - S_i(t' - t_2))]^+ - \\ & \quad (D_i(t_1) + [\max_{t'>t_1}(D_i(t') - D_i(t_1) - S_i(t' - t_1))]^+). \end{aligned} \tag{22}$$

For simplicity of notation let $b(t) = \max_{t'>t}(D_i(t') - D_i(t) - S_i(t' - t))$. We consider the following four cases.

Case 1. ($b_i(t_2) \leq 0$, $b_i(t_1) \leq 0$.) Then, from Eq. (22) and Lemma 3, we have

$$E_i(t_2) - E_i(t_1) = D_i(t_2) - D_i(t_1) \geq 0. \tag{23}$$

Case 2. ($b_i(t_2) \geq 0$, $b_i(t_1) \leq 0$.) Since $E_i(t_2) \geq D_i(t_2)$, and $E_i(t_1) = D_i(t_1)$, this case reduces to the previous one.

Case 3. ($b_i(t_2) \leq 0$, $b_i(t_1) \geq 0$.) From here, we have

$$E_i(t_2) = D_i(t_2) \geq D_i(t_2) + b_i(t_2) = \max_{t'>t_2}(D_i(t') - S_i(t' - t_2)), \tag{24}$$

and

$$E_i(t_1) = D_i(t_1) + b_i(t_1) = \max_{t'>t_1}(D_i(t') - S_i(t' - t_1)). \tag{25}$$

Let t_{max} be the value of t' that maximizes $D_i(t') - S_i(t' - t_1)$. Hence

$$E_i(t_1) = D_i(t_{max}) - S_i(t_{max} - t_1). \tag{26}$$

Now, by replacing $t' \rightarrow t_{max} + t_2 - t_1$ in Eq. (24) we obtain

$$\begin{aligned} E_i(t_2) &\geq \max_{t'>t_2}(D_i(t') - S_i(t' - t_2)) \\ &\geq D_i(t_{max} + t_2 - t_1) - S_i(t_{max} - t_2). \end{aligned} \tag{27}$$

Finally, by combining Eqs. (26) and (27), we have

$$\begin{aligned} E_i(t_2) - E_i(t_1) &\geq D_i(t_{max} + t_2 - t_1) - S_i(t_{max} - t_2) + S_i(t_{max} - t_1) - D_i(t_{max}) \quad (28) \\ &\geq D_i(t_{max} + t_2 - t_1) - D_i(t_{max}), \end{aligned}$$

where the last inequality follows from the fact that $t_{max} - t_1 > t_{max} - t_2$ and $S_i(\cdot)$ is non-decreasing. Further, since as shown in Lemma 3 $D_i(\cdot)$ is non-decreasing the proof of this case follows.

Case 4. ($b_i(t_2) \geq 0$, $b_i(t_1) \geq 0$.) The proof is practically identical to the one of the previous case. The only difference is that now $E_i(t_2)$ is strictly equal to $D_i(t_2) + b_i(t_2)$. □

The next lemma shows that the difference between the deadline values at two moments in time during an interval when a session is active, is bounded by the difference between the values in the same points of its (translated) service curve.

Lemma 5 *Consider session i characterized by the service curve $S_i(\cdot)$. Then, for any time interval $[t_1, t_2)$ while session i is active, there exists the times $t', t'' \leq t_1$ such that*

$$D_i(t_2) - D_i(t_1) \leq S_i(t_2 - t') - S_i(t_1 - t'), \tag{29}$$

and,

$$D_i(t_2) - D_i(t_1) \geq S_i(t_2 - t'') - S_i(t_1 - t''). \tag{30}$$

Proof. (Ineq. (29)) Let t' be the latest time no larger than t_1 when, by updating the deadline curve $D_i(\cdot)$, the value $D_i(t_1)$ changes. From the algorithm in Figure 6 and Eq. (7) it is easy to see that this happens only if session i becomes active at time t', and if the old value of $D_i(t_1)$ is larger than $S_i(t_1 - t') + w_i(t')$. Then, after updating $D_i(\cdot)$ at time t', we have

$$D_i(t_1) - w_i(t') = S_i(t_1 - t'), \tag{31}$$

and similarly

$$D_i(t_2) - w_i(t') = S_i(t_2 - t'). \tag{32}$$

Since between t' and t_1 the value of $D_i(t_1)$ does not change, and since the value of $D_i(t_2)$ can only decrease if updated between t' and t_2 (see Eq. (7)), by combining Eqs. (31) and (32) we have

$$D_i(t_2) - D_i(t_1) \leq S_i(t_2 - t') - S_i(t_1 - t'). \tag{33}$$

(Ineq. (30)) Let t'' be the latest time no larger than t_1 when, by updating the deadline curve $D_i(\cdot)$, the value $D_i(t_2)$ changes. Then, similarly to the previous case we have

$$D_i(t_2) - w_i(t'') = S_i(t_2 - t''). \tag{34}$$

From the definition of the deadline curve (see Eq. (7)), at time t'' we also have

$$D_i(t_1) - w_i(t'') \leq S_i(t_1 - t''). \tag{35}$$

Finally, according to the same Eq. (7), if $D_i(\cdot)$ is modified at a latter time (i.e., between t'' and t_1), then $D_i(t_1)$ can only decrease. Hence, by combining Eq. (34) and Ineq. (35), we finally obtain

$$D_i(t_2) - D_i(t_1) \geq S_i(t_2 - t'') - S_i(t_1 - t''). \tag{36}$$

□

The next three results prepares the ground for proving the first main result of this section. Theorem 1 shows that as long as the deadlines of all packets of a session are met, then its service curve is also guaranteed. Further, Lemmas 6 and 7 bounds the maximum service time requested by any session over a time interval while it is active.

Theorem 1 *The service curve of a session is guaranteed, if each of its packets is transmitted before its deadline.*

Proof. Assume this is not true. Let $w_i(t, t')$ be the service time received by session i during the time interval $[t, t')$, and let $w_i(t)$ be the *total* service time received by session i by time t. Let s be the earliest departure time of a packet of session i when its service curve is violated. Then, for any time $t < s$ when session i is passive we have $w_i(t, s) < S_i(s - t)$

(otherwise, according to Definition of the service curve, $S_i(\cdot)$ is satisfied at time s). Let s_1 be the latest time no larger than s when by updating the deadline curve $D_i(\cdot)$, the value $D_i(s)$ changes. From the algorithm in Figure 6 and Eq. (7) it is easy to see that this happens only if session i becomes active at time s_1, and if the old value of $D_i(s)$ is larger than $S_i(t - s_1)$. Then, after updating $D_i(\cdot)$, we have

$$D_i(s) - w_i(s_1) = S_i(s - s_1). \tag{37}$$

Next note that the deadline of the packet sent at time s is simply $D_i^{-1}(w_i(s))$ (see the algorithm in Figure 6). Since we assume that all deadlines are satisfied, it follows that $D_i^{-1}(w_i(s)) \geq s$. From here, we have

$$w_i(s) \geq D_i(s). \tag{38}$$

Finally, since $w_i(s_1, s) = w_i(s) - w_i(s_1)$, from Eq. (37) and Ineq. (38) it follows that $w_i(s, s_1) \geq S_i(s - s_1)$, which concludes the proof. $\square$

The next lemma gives an upper bound on the service that a session may request over any future time interval. The service requested by a session is defined formally as follows.

Definition 3 *The total service requested by session i at time t (when it is active) is $D_i(t)$.*

Lemma 6 *Let $w_i(t)$ be the total service time received by session i by time t. Then, the maximum service time requested by session i over any future interval $[t, t_1)$ is achieved when the session is continuously backlogged during this interval.*

Proof. Consider two cases, whether session i is passive or not at time t.

Case 1. (session i is active at time t) Then the additional service time requested by session i between t and t_1, denoted $r_i(t, t_1)$, is bounded as follows

$$r_i(t, t_1) \leq D_i(t_1) - w_i(t), \tag{39}$$

where the equality holds when session i has a packet with the deadline t_1.

Next, assume that session i becomes passive at some time $t' \leq t_1$. Then, we have two sub-cases, whether the session becomes again active before t_1, or not. If it becomes again active, then from Eq. (7), it is easy to see that the new value of $D_i(t_1)$ cannot increase. On the other hand, if the session remains passive by time t_1, then the total requested service time over the interval $[t, t_1)$ is bounded as follows

$$r_i(t, t_1) \le D_i(t') - w_i(t) \le D_i(t_1) - w_i(t), \tag{40}$$

where the last inequality follows from the fact that $D_i(\cdot)$ is a non-decreasing function. Thus, we have shown that if at any point between t and t_1 session i becomes passive, then $D_i(t_1)$ does not increase, and consequently the upper bound of the maximum service time requested by session i over the interval $[t, t_1)$ will not increase.

Case 2. (session i is passive at time t) Let $p_1(t', t)$ denote the value of $D_i(t)$ if session i becomes active at time t', and remains continuously backlogged until time t. Next, from Eq. 7, it is easy to see that $p_i(t', t_1)$ is a non-decreasing function in t'. Therefore $p_i(t', t)$, and consequently $D_i(t_1)$ is maximized when session i becomes active at time t^+, and remains continuously backlogged until t_1. □

Lemma 7 *Let i be a sessions that has no eligible request at time t. Then the service requested over any future interval $[t, t_1)$ is no larger than $S_i(t_1 - t)$.*

Proof. We consider two cases, whether session i is active at time t, or not.

Case 1. (session i is active at time t.) Let $w_i(t)$ be the service time received by session i by time t. Since the session is not eligible at time t it follows that:

$$E_i(t) > w_i(t). \tag{41}$$

Since the maximum service requested by session i by time t_1 is $D_i(t_1)$, it follows that between t and t_1 session i requests at most an additional $D_i(t_1) - E_i(t)$ service. By substituting $E_i(t)$, and using Ineq. (41), we have

$$\begin{aligned} D_i(t_1) - w_i(t) &< D_i(t_1) - E_i(t) \\ &= D_i(t_1) - D_i(t) - [\max_{t'>t}(D_i(t') - D_i(t) - S_i(t'-t))]^+ \\ &\le D_i(t_1) - D_i(t) - [D_i(t_1) - D_i(t) - S_i(t_1 - t)]^+. \end{aligned} \tag{42}$$

Further, note that for any two reals x and y, we have $x - [x-y]^+ \le y$. By substituting $x \leftarrow (D_i(t_1) - D_i(t))$ and $y \leftarrow S_i(t_1 - t)$, respectively, we finally obtain

$$D_i(t_1) - w_i(t) < S_i(t_1 - t). \tag{43}$$

Case 2. (session i is passive at time t) Assume that i becomes active at time t', such that $t \le t' \le t_1$. (If $t' > t_1$, then session i does not request any service time during the interval $[t, t_1)$, and therefore the lemma is trivially true.) Then, since $w_i(t) = w_i(t')$, and from Eq. (7) it follows that

$$D_i(t_1) - w_i(t) = D_i(t_1) - w_i(t') \le S_i(t_1 - t') \le S_i(t_1 - t). \tag{44}$$

which concludes the proof. □

Theorem 2 *The H-FSC algorithm guarantees that the deadline of any packet is not missed by more than τ_{max}, where τ_{max} represents the time to transmit a packet of maximum length.*

Proof. Assume the deadline, d_l^k, of the k-th packet of session l is missed by more than τ_{max}. Let B denote the set of all sessions that prior to time d_l^k have at least a deadline *larger* than d_l^k. Then, let t_1 be the largest time no greater than d_l^k, when a session in B receives service time, if any. Similarly, let t_2 denote the largest time no greater than d_l^k, when the server is idle. Finally take $s = \max(t_1, t_2)$.

Let $\mathcal{A}_1(s)$ denote the set of all sessions active at time s that have a deadline smaller than d_l^k. Clearly, there is no session in $\mathcal{A}_1(s)$ having an eligible request at time s. (Otherwise, no session in B can be served at time s, or alternatively, the server cannot be idle.) Consequently, we have

$$E_i(s) \ge D_i(s), \text{ for any } i \in \mathcal{A}_1(s). \tag{45}$$

Then it is easy to see that *only* the sessions in $\mathcal{A}_1(s)$, and eventually new sessions that become active *after* time s can be served before d_l^k. Let C denote the set of all these sessions. Then, according to Lemma 7 the maximum service time requested by any of these sessions over the interval $[s, d_l^k)$ is no larger than $S_i(d_l^k - l)$. Consequently, the total service time requested by *all* sessions during the interval $[s, d_l^k)$ is no larger than

$$\sum_{i \in C} S_i(d_l^k - s) \leq d_l^k - s, \tag{46}$$

where here, for simplicity, we consider a server with rate 1. Next, note that the transmission of the packet served at time s is completed no latter than $s + \tau_{max}$. Since between this time and time $d_l^k + \tau_{max}$ only the packets of the sessions in C are served, it follows that the sessions in C receive at least $d_l^k + \tau_{max} - (s + \tau_{max}) = d_l^k - s$ service time. Since as shown by Eq. (46), the sum of *all* requests with the deadline no larger than d_l^k is no larger than $d_l^k - s$, it follows that the request of session l is served as well (i.e., its packet is transmitted by time $d_l^k - s$), which contradicts the hypothesis, and therefore proves the theorem. □

The next six lemmas are used to prove Theorem 3, i.e., the difference between the virtual times of two simultaneously active leaf classes (sessions) is bounded. Lemma 8 bounds the total service $c_i(t)$ received by session i each time it was selected based on its eligible time and deadline, while Lemma 10 (using the results of Lemma 9) bounds the same service time received by session i over an arbitrary interval of time while session is active. Next, Lemmas 11 and 12 provide bounds for a session's virtual curve in terms of its service curve. Finally, Lemma 13 bounds the *increase* in the discrepancy between the virtual times of two leaf classes while they are simultaneously active.

Lemma 8 *For any time t when session i is active we have*

$$D_i(t - \tau_{max}) - l_{i,max} \leq c_i(t) \leq E_i(t) + l_{i,max}, \tag{47}$$

where τ_{max} represents the time to transmit the packet of maximum size, and $l_{i,max}$ represents the maximum size of a packet of session i.

Proof. ($D_i(t - \tau_{max}) - l_{i,max} \leq c_i(t)$.) Assume this is not true, i.e.,

$$D_i(t - \tau_{max}) > c_i(t) + l_{i,max} \tag{48}$$

Let d be the deadline of the packet at the head of the queue, and let l_i be its length. Then, according to the algorithm in Figure 6, we have $d = D_i^{-1}(c_i(t) + l_i)$. Since D_i^{-1} is non-decreasing, it follows that $d \leq D_i^{-1}(c_i(t) + l_{i,max})$. On the other hand, from Ineq. (48) we have $t - \tau_{max} > D_i^{-1}(c_i(t) + l_{i,max})$. From here it follows that

$$d + \tau_{max} < t. \tag{49}$$

Thus, at time t the deadline of the packet at the head of the queue of session i is already missed by more than $d + \tau_{max}$, which contradicts Theorem 2 and therefore proves this case.

($c_i(t) \leq E_i(t) + l_{i,max}$.) Let t_1 be the latest time no larger than t when a packet of session i is scheduled, and let l_i be the length of this packet. Then, at time t_1, we have $E_i(t_1) \geq c_i(t_1)$, and further $c_i(t) = c_i(t_1) + l_i \leq c_i(t_1) + l_{i,max} \leq E_i(t_1) + l_{i,max}$. Since $E_i(\cdot)$ is non-decreasing (see Lemma 4) the proof follows. $\square$

Lemma 9 *For any times t_1 and t_2, such that $t_1 \leq t_2$, and for any session i characterized by the service curve $S_i(\cdot)$, we have*

$$S_i(t_2 - t_1) - B^{S_i}(t_2 - t_1) \leq E_i(t_2) - D_i(t_1) \leq S_i(t_2 - t_1) + B^{S_i}_{max} + B^{S_i}(t_2 - t_1). \tag{50}$$

Proof. From the definition of the eligible curve (Eq. (10)) we have

$$E_i(t_2) - D_i(t_1) = D_i(t_2) + [\max_{t' > t_2}(D_i(t') - D_i(t_2) - S_i(t' - t_2))]^+ - D_i(t_1). \tag{51}$$

We consider two cases: (1) $\max_{t'>t_2}(D_i(t') - D_i(t_2) - S_i(t' - t_2)) \geq 0$, and (2) $\max_{t'>t_2}(D_i(t') - D_i(t_2) - S_i(t' - t_2)) < 0$.

Case 1. From Eq. (51) we have

$$\begin{aligned} E_i(t_2) - D_i(t_1) &= D_i(t_2) + \max_{t'>t_2}(D_i(t') - D_i(t_2) + S_i(t' - t_2)) - D_i(t_1) \qquad (52) \\ &= \max_{t'>t_2}(D_i(t') - S_i(t' - t_2)) - D_i(t_1) \\ &= \max_{t'>t_2}(D_i(t') - D_i(t_1) - S_i(t' - t_2)). \end{aligned}$$

Further, from Lemma 5 it follows that there exists a time $t_0 \leq t_1$, such that $D_i(t') - D_i(t_1) \leq S_i(t' - t_0) - S_i(t_1 - t_0)$. From here, and by using Lemmas 1, 2 and Definition 2, we obtain

$$\begin{aligned} E_i(t_2) - D_i(t_1) &\leq \max_{t'>t_2}(S_i(t' - t_0) - S_i(t_1 - t_0) - S_i(t' - t_2)) \qquad (53) \\ &= \max_{t'>t_2}(S_i((t_2 - t_0) + t' - t_2) - S_i(t_1 - t_0) - S_i(t' - t_2)) \end{aligned}$$

$$
\begin{aligned}
&\leq \max_{t'>t_2}(S_i(t_2-t_0)+S_{i,max}(t'-t_2)-S_i(t_1-t_0)-S_i(t'-t_2)) \quad \text{(Lemma 1)}\\
&= S_i(t_2-t_0)-S_i(t_1-t_0)+\max_{t'>t_2}(S_{i,max}(t'-t_2)-S_i(t'-t_2))\\
&\leq S_i(t_2-t_0)-S_i(t_1-t_0)+B^{S_i}_{max} \quad \text{(Definition 2)}\\
&\leq S_i(t_2-t_1)+B^{S_i}(t_2-t_1)+B^{S_i}_{max} \quad \text{(Lemma 2).}
\end{aligned}
$$

Case 2. In this case, Eq. (51) reduces to

$$E_i(t_2)-D_i(t_1)=D_i(t_2)-D_i(t_1). \tag{54}$$

Similar to the previous case, from Lemma 5 it follows that there exists a time $t_0 \leq t_1$, such that $D_i(t_2)-D_i(t_1) \geq S_i(t_2-t_0)-S_i(t_1-t_0)$. Further by using Lemma 2, we have

$$D_i(t_2)-D_i(t_1) \geq S_i(t_2-t_0)-S_i(t_1-t_0) \geq S_i(t_2-t_1)-B^{S_i}(t_2-t_1). \tag{55}$$

Since the proof for the left-hand side of Ineq. (50) is similar to the previous proof we omit it here. □

Lemma 10 *Consider session i characterized by the service curve $S_i(\cdot)$. Then, for any interval $[t_1,t_2)$ while session i is active, we have*

$$S_i(t_2-t_1)-\theta_i(t_2-t_1) \leq c_i(t_1,t_2) \leq S_i(t_2-t_1)+\theta_i(t_2-t_1)+B^{S_i}_{max}, \tag{56}$$

where

$$\theta_i(t)=B^{S_i}(t)+S_{i,max}(\tau_{max})+2l_{i,max}. \tag{57}$$

Proof. By applying Lemma 8 for t_1 and t_2 we get

$$
\begin{aligned}
D_i(t_1-\tau_{max})-l_{i,max} \leq c_i(t_1) \leq E_i(t_1)+l_{i,max}, \text{ and}\\
D_i(t_2-\tau_{max})-l_{i,max} \leq c_i(t_2) \leq E_i(t_2)+l_{i,max}
\end{aligned} \tag{58}
$$

Since $c_i(t_1,t_2)=c_i(t_2)-c_i(t_1)$, from here we have

$$D_i(t_2-\tau_{max})-E_i(t_1)-2l_{i,max} \leq c_i(t_1,t_2) \leq E_i(t_2)-D_i(t_1-\tau_{max})+2l_{i,max}. \tag{59}$$

For the left-hand side inequality we have

$$
\begin{aligned}
c_i(t_1, t_2) &\geq D_i(t_2 - \tau_{max}) - E_i(t_1) - 2l_{i,max} \\
&= D_i(t_2 - \tau_{max}) - D_i(t_1) + D_i(t_1) - E_i(t_1) - 2l_{i,max} \\
&\geq D_i(t_2 - \tau_{max}) - D_i(t_1) - 2l_{i,max},
\end{aligned} \tag{60}
$$

where the last inequality follows from the fact that $D_i(t)$ is never larger than $E_i(t)$ (see Eq. 7). From Lemma 5 it follows that there exists a time $t'' \leq \min(t_1, t_2 - \tau_{max})$, such that

$$
\begin{aligned}
c_i(t_1, t_2) &\geq S_i(t_2 - \tau_{max} - t'') - S_i(t_1 - t'') - 2l_{i,max} \\
&\geq S_i(t_2 - \tau_{max} - t_1) - B^{S_i}(t_2 - \tau_{max} - t_1) - 2l_{i,max} \\
&\geq S_i(t_2 - t_1) - S_{i,max}(\tau_{max}) - B^{S_i}(t_2 - \tau_{max} - t_1) - 2l_{i,max} \\
&\geq S_i(t_2 - t_1) - S_{i,max}(\tau_{max}) B^{S_i}(t_2 - t_1) - 2l_{i,max},
\end{aligned} \tag{61}
$$

where the second inequality follows from Lemma 2, and the third inequality follows from Definition 2.

Next, by applying again Lemmas 9 and 2 we prove the right-hand side of Ineq. (59).

$$
\begin{aligned}
c_i(t_1, t_2) &\leq E_i(t_2) - D_i(t_1 - \tau_{max}) + 2l_{i,max} \\
&\leq S_i(t_2 - t_1 + \tau_{max}) + B^{S_i}_{max} + B^{S_i}(t_2 - t_1) + 2l_{i,max} \\
&\leq S_i(t_2 - t_1) + S_{i,max}(\tau_{max}) + B^{S_i}_{max} + B^{S_i}(t_2 - t_1) + 2l_{i,max}.
\end{aligned} \tag{62}
$$

□

Lemma 11 *For any session i characterized by the service curve $S_i(\cdot)$, and for any interval $[t, t + \Delta t]$ while the session is active, we have*

$$
S_i(\Delta t) - B^{S_i}(\Delta t) \leq V_i(t + \Delta t) - V_i(t) \leq S_i(\Delta t) + B^{S_i}(\Delta t). \tag{63}
$$

Proof. ($V_i(t + \Delta t) - V_i(t) < S_i(\Delta t) + B^{S_i}(\Delta t)$.) Let t_1 be the latest time no larger than t when the value of $V_i(t)$ has been *modified*. Then, according to Eq. (11) we have $V_i(t) = S_i(t - v_i) + w_i(t_1)$, where v_i represents the average between the maximum and the minimum virtual times of any two sessions active at time t_1, if any, and $w_i(t_1)$ represents the total service time received by session i so far. Consequently, the value of $V_i(t+\Delta t)$ computed at time t_1 is at most $S_i(t+\Delta t - v_i) + w_i(t_1)$. Since from the definition of the deadline curve its value cannot increase when it is eventually again updated after t_1 (see Eq. (7)), it follows that

$$\begin{aligned} V_i(t+\Delta t) - V_i(t) &= V_i(t+\Delta t) - S_i(t-v_i) - w_i(t_1) \\ &\leq S_i(t+\Delta t - v_i) + w_i(t_1) - S_i(t-v_i) - w_i(t_1) \\ &\leq S_i(t+\Delta t - v_i) - S_i(t-v_i) \\ &\leq S_i(\Delta t) + B^{S_i}(\Delta t), \end{aligned} \tag{64}$$

where the last inequality follows from Lemma 2.

$(S_i(\Delta t) - B^{S_i}(\Delta t) < V_i(t+\Delta t) - V_i(t).)$ The proof is similar to the previous case, and therefore we do not give it here. The only difference is that in this case choose t_1 to be the latest time no larger than t when the value of $V_i(t+\Delta t)$ is updated. □

Lemma 12 *For any session i characterized by the service curve $S_i(\cdot)$, and for any time interval t while the session is active, we have*

$$S_i^{-1}(t) - S_{i,max}^{-1}(B_{max}^{S_i}) \leq V_i^{-1}(t) \leq S_i^{-1}(t) + S_{i,max}^{-1}(B_{max}^{S_i}). \tag{65}$$

Proof. We give the proof for the left-hand side inequality. (The proof for the right-hand side inequality is identical.) From Lemma 11, for any $x \geq 0$, we have

$$V_i(x) \leq S_i(x) + B^{S_i}(x) \leq S_i(x) + B_{max}^{S_i}. \tag{66}$$

Next by substituting $t = V_i(x)$ in the above equation, we have

$$t \leq S_i(V_i^{-1}(t)) + B_{max}^{S_i} \Rightarrow S_i^{-1}(t - B_{max}^{S_i}) \leq V_i^{-1}(t). \tag{67}$$

Further, by using Lemma 1 the proof follows. □

Lemma 13 *For any two sibling sessions (leaf classes) i and j that are simultaneously active during the interval $[t_1, t_2)$, the difference between their virtual times at any time $t \in [t_1, t_2)$ is bounded as follows*

$$\min(v_i(t_1) - v_j(t_1), 0) - \delta_{j,i} \leq v_i(t) - v_j(t) \leq \max(v_i(t_1) - v_j(t_1), 0) + \delta_{i,j}, \tag{68}$$

where

$$\begin{aligned} \delta_{i,j} &= S_{i,max}^{-1}(\theta_{i,max} + B_{max}^{S_i} + l_{i,max}) + S_i^{-1}(B_{max}^{S_i}) + S_{j,max}^{-1}(\theta_j) + S_j^{-1}(B_{max}^{S_j}), \quad (69) \\ \theta_{i,max} &= \max_{t \geq 0} \theta_i(t). \end{aligned}$$

Proof. Let t_0 be the largest time, if any, in the interval $[t_1, t)$ when session i is selected based on its virtual time. We consider two cases whether time t_0 exists or not.

Case 1. (t_0 exists.) If session i was selected at time t_0 based on its virtual time, then the followings are true: (1) neither session i nor session j have an eligible packet, i.e., $c_i(t_0) > E_i(t_0)$ and $c_i(t_0) > E_i(t_0)$, and (2) the virtual time of session i is smaller than the one of session j, i.e., $v_i(t_0) \leq v_j(t_0)$. Since after that and until time t session i is never served based on its virtual time, the total service receive by session i during the interval $[t_0, t]$ is

$$w_i(t_0, t) = c_i(t_0, t) + l_i, \tag{70}$$

where l_i is the length of the packet served at time t_0. On the other hand, the service time received by session j during the same time interval is

$$w_j(t_0, t) \geq c_j(t_0, t), \tag{71}$$

Now, note that

$$v_i(t) = V_i^{-1}(w_i(t)) = V_i^{-1}(w_i(t_0) + w_i(t_0, t)) = V_i^{-1}(w_i(t_0) + c_i(t_0, t) + l_i), \tag{72}$$

and

$$v_j(t) = V_j^{-1}(w_j(t)) \geq V_j^{-1}(w_j(t_0) + c_j(t_0, t)). \tag{73}$$

Further, by using Lemmas 1, 10, and 12, we have

$$
\begin{aligned}
V_i^{-1}(w_i(t_0)+c_i(t_0,t)+l_i) &\le V_i^{-1}(w_i(t_0)) + V_{i,max}^{-1}(c_i(t_0,t)+l_i) \\
&= v_i(t_0) + V_{i,max}^{-1}(c_i(t_0,t)+l_i) \\
&\le v_i(t_0) + V_{i,max}^{-1}(S_i(t-t_0)+\theta_{i,max}+B_{max}^{S_i}+l_i) \\
&\le v_i(t_0) + S_i^{-1}(S_i(t-t_0)+\theta_{i,max}+B_{max}^{S_i}+l_i) + S_{i,max}^{-1}(B_{max}^{S_i}) \\
&\le v_i(t_0) + S_i^{-1}(S_i(t-t_0)) + S_{i,max}^{-1}(\theta_{i,max}+B_{max}^{S_i}+l_i) + S_{i,max}^{-1}(B_{max}^{S_i}) \\
&\le v_i(t_0) + t - t_0 + S_{i,max}^{-1}(\theta_{i,max}+B_{max}^{S_i}+l_i) + S_{i,max}^{-1}(B_{max}^{S_i}) \\
&\le v_i(t_0) + t - t_0 + S_{i,max}^{-1}(\theta_{i,max}+B_{max}^{S_i}+l_{i,max}) + S_{i,max}^{-1}(B_{max}^{S_i}).
\end{aligned} \tag{74}
$$

and

$$
\begin{aligned}
V_j^{-1}(w_j(t_0)+c_j(t,t_0)) &\ge V_j^{-1}(w_j(t_0)) + V_{j,min}^{-1}(c_j(t,t_0)) \\
&\ge V_j^{-1}(w_j(t_0)) + V_{j,min}^{-1}(S_j(t-t_0)-\theta_j) \\
&= v_j(t_0) + V_{j,min}^{-1}(S_j(t-t_0)-\theta_j) \\
&= v_j(t_0) + S_j^{-1}(S_j(t-t_0)-\theta_j) - S_{j,max}^{-1}(B_{max}^{S_j}) \\
&\ge v_j(t_0) + S_j^{-1}(S_j(t-t_0)) - S_{j,max}^{-1}(\theta_j) - S_{j,max}^{-1}(B_{max}^{S_j}) \\
&= v_j(t_0) + (t-t_0) - S_{j,max}^{-1}(\theta_j) - S_{j,max}^{-1}(B_{max}^{S_j}).
\end{aligned} \tag{75}
$$

Since $v_i(t_0) \le v_j(t_0)$, by combining Eqs. (72), (73), (74) and (75) we obtain

$$
v_i(t) - v_j(t) \le S_{i,max}^{-1}(\theta_{i,max}+B_{max}^{S_i}+l_{i,max}) + S_i^{-1}(B_{max}^{S_i}) + S_{j,max}^{-1}(\theta_j) + S_j^{-1}(B_{max}^{S_j}). \tag{76}
$$

Case 2. (t_0 does not exist.) This case is very similar to the previous one. The only difference is that we compute the differences between the virtual times over the interval $[t_1, t)$, instead $[t_0, t)$. $\square$

Theorem 3 *The difference between the virtual times of any two sibling sessions i and j that are simultaneously active at time t is bounded as follows*

$$
-\delta_{max} - \delta_{j,i} \le v_i(t) - v_j(t) \le \delta_{max} + \delta_{i,j}, \tag{77}
$$

where

$$
\delta_{max} = \max_{i,j \in \mathcal{A}(t)} \delta_{i,j}. \tag{78}
$$

Proof. The proof is by induction on the moments of time t_0, t_1, ..., when a session become active. Without loss of generality we assume that at any time only one session may become active.[5]

Basic Step. Since the first session becomes active at time t_0 and since between t_0 and t_1 this is the only active session, Ineq. (77) is trivially true.

Induction Step. Consider an arbitrary session i that becomes active at time t_k, and let t_{k+1} be the time when the next session becomes active. Let j_1 and j_2 be the active sessions with the minimum, respectively the maximum virtual time at time t_k, i.e.,

$$v_{j_2}(t_k) - v_{j_1}(t_k) = \max_{l,m \in \mathcal{A}(t)} (v_l(t_k) - v_m(t_k)). \tag{79}$$

According to the induction hypothesis

$$v_{j_2}(t_k) - v_{j_1}(t_k) \leq \delta_{max} + \delta_{j_1,j_2} \leq 2\delta_{max}. \tag{80}$$

Recall that when session i joins the competition at time t_k its virtual time is initialized to $v_i(t_k) = (v_{j_2}(t_k) - v_{j_1}(t_k))/2$. Then, from Lemma 13 it follows that for any time $t \in [t_k, t_{k+1})$, the difference between the virtual time of session i and of any other session j that is active in the entire interval $[t_k, t_{k+1})$ is bounded as follows

$$\begin{aligned} v_i(t) - v_j(t) &\leq \max(v_i(t_k) - v_j(t_k), 0) + \delta_{i,j} \\ &= \max\left(\frac{v_{j_2}(t_k) + v_{j_1}(t_k)}{2} - v_j(t_k), 0\right) + \delta_{i,j}. \end{aligned} \tag{81}$$

Since $v_{j_1}(t_k) \leq v_j(t_k)$, further we have

$$\begin{aligned} v_i(t) - v_j(t) &\leq \max\left(\frac{v_{j_2}(t_k) - v_{j_1}(t_k)}{2}, 0\right) + \delta_{i,j} \\ &\leq \delta_{max} + \delta_{i,j}. \end{aligned} \tag{82}$$

To get the inferior bound, we proceed similarly

[5] If there are two or more sessions that become active simultaneously we simply consider that they become active at t, $t + \epsilon$, $t + 2\epsilon$, ..., and make $\epsilon \to 0$.

$$
\begin{aligned}
v_i(t) - v_j(t) &\geq \min(v_i(t_k) - v_j(t_k), 0) - \delta_{j,i} \\
&= \min\left(\frac{v_{j_2}(t_k) + v_{j_1}(t_k)}{2} - v_j(t_k), 0\right) - \delta_{j,i} \\
&\geq \min\left(-\frac{v_{j_2}(t_k) - v_{j_1}(t_k)}{2}, 0\right) - \delta_{j,i} \\
&= -\frac{v_{j_2}(t_k) - v_{j_1}(t_k)}{2} - \delta_{j,i} \\
&\geq -\delta_{max} - \delta_{j,i}.
\end{aligned} \tag{83}
$$

□

Since for any linear function $f(\cdot)$ we have $B^f_{max} = 0$, from the previous theorem and Lemmas 10 and 13 we have the following result.

Corollary 1 *Consider two sibling sessions (leaf classes) i and j characterized by linear service curves with slopes r_i and r_j, respectively, that are simultaneously active during the interval $[t_1, t_2)$. Assume that the switch capacity is C. Then, the difference between their virtual times at any time $t \in [t_1, t_2)$ is bounded as follows*

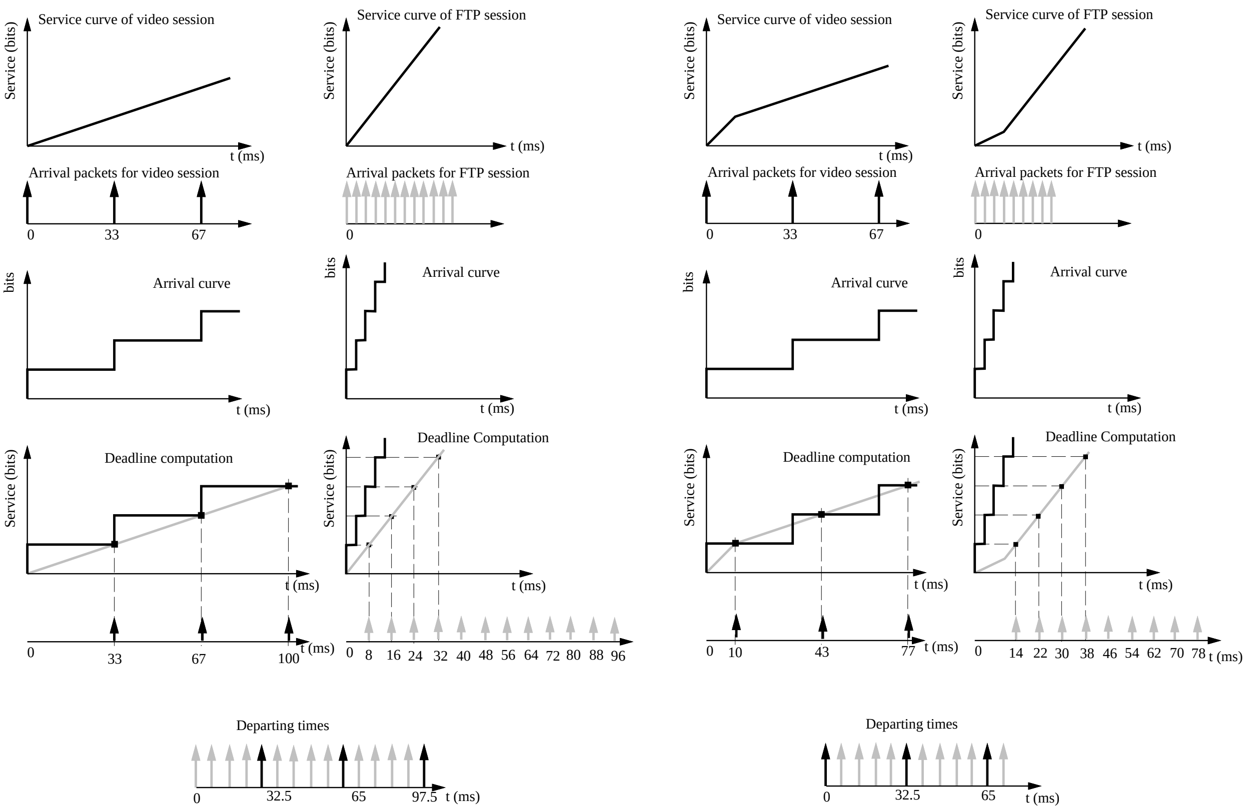
$$
-\delta_{max} - \delta_{j,i} \leq v_i(t) - v_j(t) \leq \delta_{max} + \delta_{i,j}, \tag{84}
$$

where

$$
\delta_{i,j} = 3\frac{l_{i,max}}{r_i} + 2\frac{l_{j,max}}{r_j} + \frac{l_{max}}{C}, \tag{85}
$$

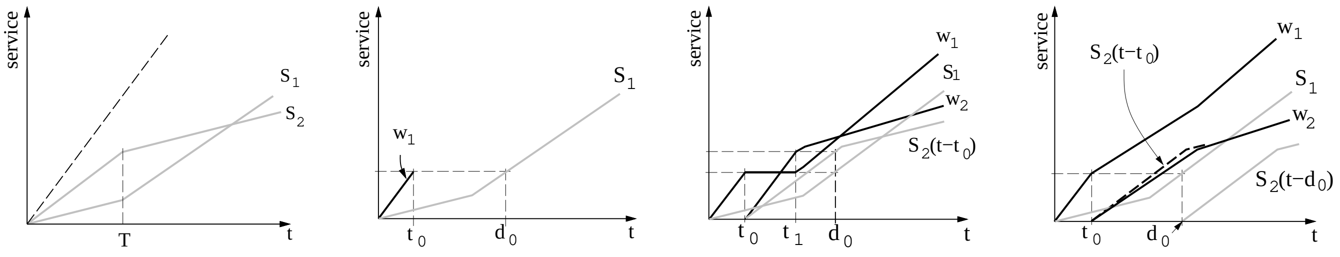
and

$$
\delta_{max} = l_{max}\left(\frac{3}{r_i} + \frac{2}{r_j} + \frac{1}{C}\right). \tag{86}
$$



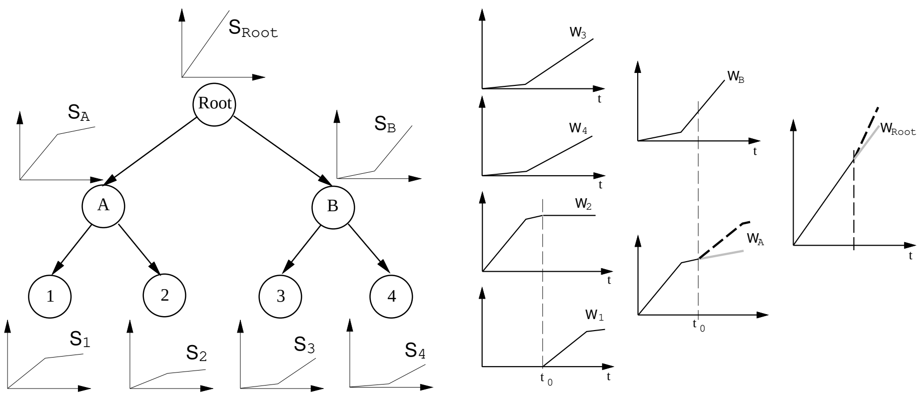
(a) (b)

Figure 2: *An example illustrating the benefits of delay-bandwidth decoupling. The video session requires a bandwidth of 2 MBps and has a delay target of 10 ms. The FTP session requires 8 Mbps. The total capacity of the link is 10 Mbps. (a) The service curves and the resulting schedule when only bandwidth is used to specify the sessions' requirements. The delay of the video packets is over 26 ms. (b) The service curves and the resulting schedule when delay and bandwidth are both specified for each session. The delay of the video packets is now less than 10 ms.*



(a) (b) (c) (d)

Figure 3: *An example illustrating the "punishment" of a session under SCED policy: (c) session 1 does not receive any service during* $(t_0, t_1]$*, after session 2 becomes active at* t_0*. (d) A modified version of SCED that tries to not penalize session 1 at all, but violates session 2's service curve.*



(a) (b)

Figure 4: *An example illustrating why it is not possible to guarantee the service curves of all the classes in the hierarchy. (a) The hierarchy and the service curves of each node. (b) The service received by each session when sessions 2, 3, and 4 become active at time* 0*; session 1 becomes active at time* t_0*.*

receive_packet(i,p) /* *session* i *has received packet* p */
  enqueue$(queue_i, p)$;
  if $(i \notin \mathcal{A})$ /* *if* i *was not active* */
    update_ed$(i, 0, p)$; /* *update* $E_i(\cdot)$, $D_i(\cdot)$, *compute* e_i, d_i */
    update_v(i, p); /* *update* $V(\cdot)$ *for* i *and its ancestors* */
    $\mathcal{A} = \mathcal{A} \cup \{i\}$; /* *mark* i *active* */

get_packet() /* *get next packet to send* */
  $i = \min_{d_i}\{i \in \mathcal{A} \mid e_i \leq t\}$; /* *select by real-time criteria* */
  if $(i \neq \emptyset)$ /* *does such session exists ?* */
    $p =$**dequeue**$(queue_i)$;
    update_v(i, p); /* *update virtual time* */
    if $(queue_i \neq \emptyset)$
      update_ed$(i, p, head(queue_i))$;
    else
      $\mathcal{A} = \mathcal{A} \setminus \{i\}$; /* *mark* i *passive* */
  else /* *select active ses. by link-sharing criteria* */
    $i = \min_{v_i} \mathcal{A}$;
    $p =$**dequeue**(i);
    update_v(i, p)
    if $(queue_i \neq \emptyset)$
      update_d$(i, p, head(queue_i))$ /* *update* d_i *only* */
    else
      $\mathcal{A} = \mathcal{A} \setminus \{i\}$;
  send_packet(p);

Figure 5: *The Hierarchical Fair Service Curve (H-FSC) algorithm. The* **receive_packet** *function is executed every time a packet arrives; the* **get_packet** *function is executed every time a packet departs (to select the next packet to send).*

$\mathbf{update\_ed}(i, p, next\_p)$

$\quad$ **if** $(i \notin \mathcal{A})$

$\quad\quad$ /* *session i is about to become active* /*

$\quad\quad$ $D_i(\cdot) =$**update_DC**(i); /* *update deadline curve* */

$\quad\quad$ $E_i(\cdot) =$**update_EC**(i); /* *update eligible curve* */

$\quad$ $c_i = c_i + \mathbf{length}(p)$;

$\quad$ $e_i = E_i^{-1}(c_i)$; /* *update eligible time* */

$\quad$ $d_i = D_i^{-1}(c_i + \mathbf{length}(next\_p))$; /* *update deadline* */

(a)

$\mathbf{update\_d}(i, p, next\_p)$

$\quad$ $d_i = D_i^{-1}(c_i - \mathbf{length}(p) + \mathbf{length}(next\_p))$;

(b)

Figure 6: *(a) The function which updates the deadline and the eligible curves, and computes the deadlines and the eligible times for each session. Note that the eligible and the deadline curves are updated only when the session becomes active. (b) The function which updates the virtual deadline, when the session is served by the link share criteria. This is because the new packet at the head of the queue may have a different length.*

update_v(i, p)

  $n =$ **parent**(i);

  if $(i \notin \mathcal{A})$ /* *is class/session i active ?* */

    $v_i = max(v_i, v^s_{p(i)})$

    update_VC(i);

    if (**active**(i) = *TRUE*)

      return;

  else

    $w_i = w_i +$ **length**(p);

    $v_i = V_i^{-1}(w_i)$;

  if $(n \neq ROOT)$

    update_v(n, p);

Figure 7: *The function which updates the virtual time curves and the virtual times in H-FSC.*

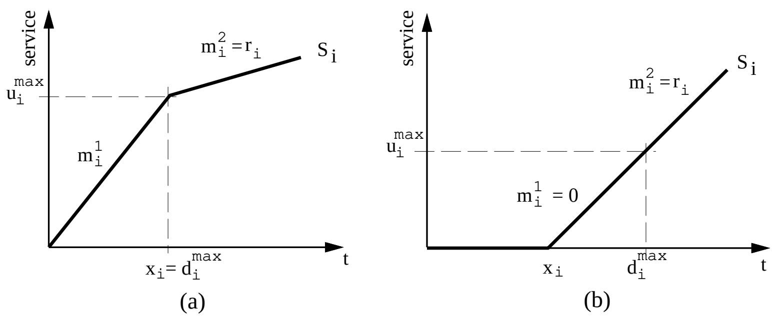


Figure 8: *The service curve associated with a session i characterized by maximum delay* d_i^{max}*, maximum unit of work* u_i^{max}*, and average rate* r_i*. If* $u_i^{max}/d_i^{max} > r_i$*, the service curve is concave (a); otherwise, it is convex (b).*

update_DC(i)

  if $((m_i^1 > m_i^2)$ **and** $(c_i + y_i^S - y_i > m_i^2 \times (t_a + x_i^S - x_i)))$

    $/*$ *$D_i(\cdot)$ concave and intersects $S_i(\cdot - t_a) + c_i$* $*/$

    $a = y_i - m_i^2 x_i;$ $/*$ *compute intersection point* $*/$

    $x_i = (c_i - m_i^1 \times (x_i^S + t_a) - a)/(m_i^2 - m_i^1);$

    $y_i = m_i^2 x_i + a;$

  else

    $x_i = t_a + x_i^S;$

    $y_i = c_i + y_i^S;$

Figure 9: *The function which updates the deadline curve D_i. (Service curve parameters are identified by superscript S.)*

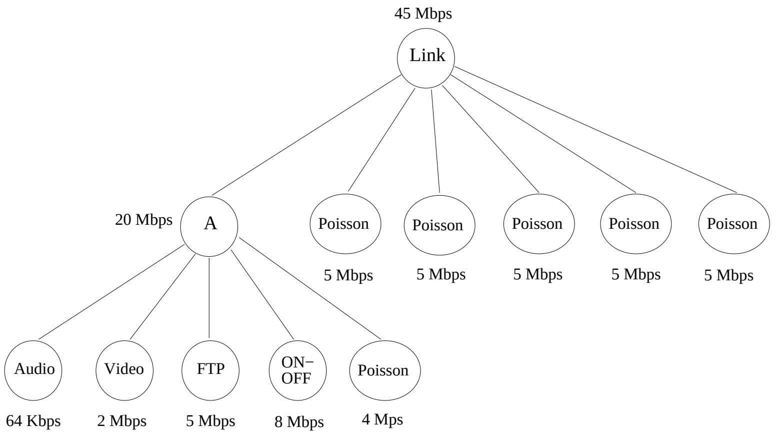
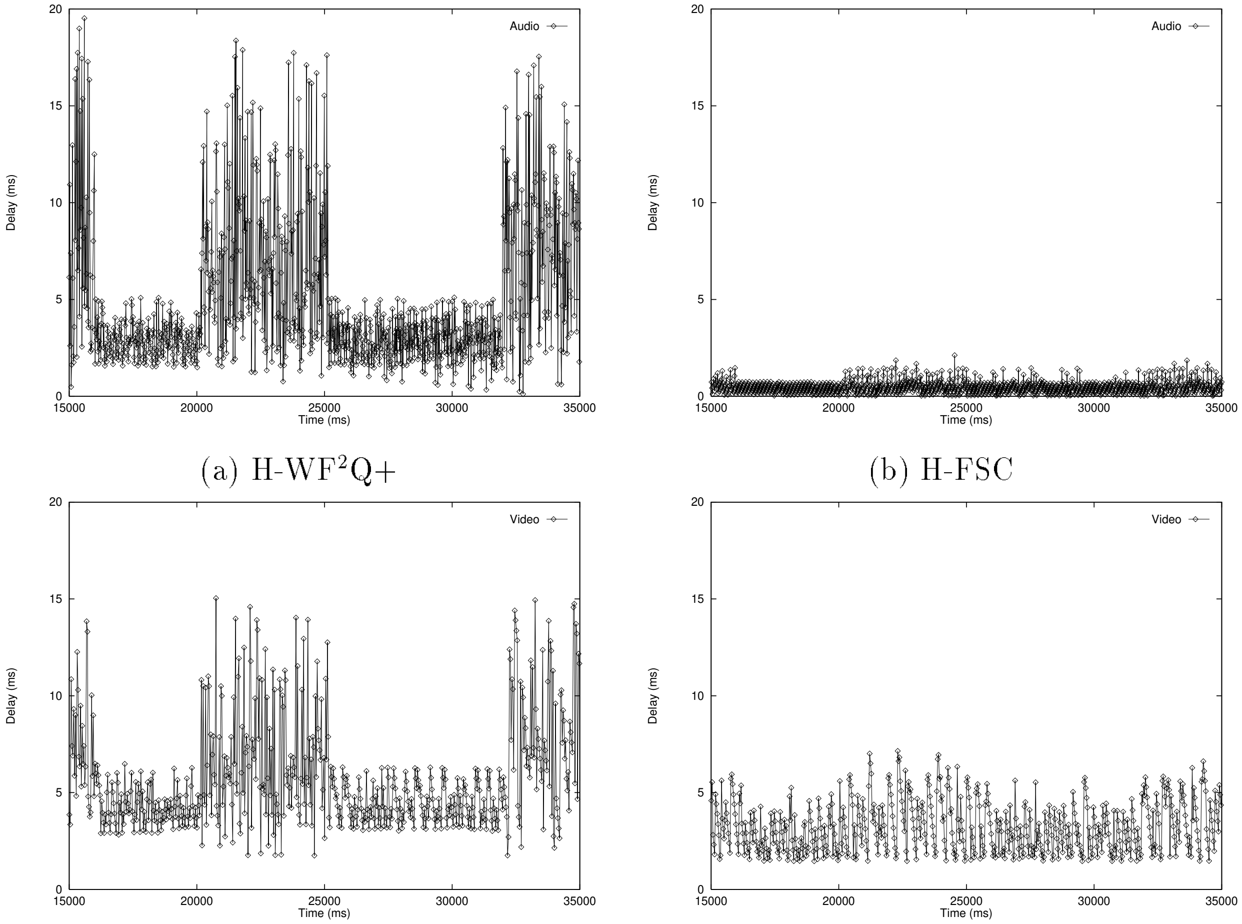


Figure 10: *Class Hierarchy.*



(a) H-WF^{2}Q+ (b) H-FSC

(c) H-WF^{2}Q+ (d) H-FSC

Figure 11: *Absolute delay for audio and video sessions.*

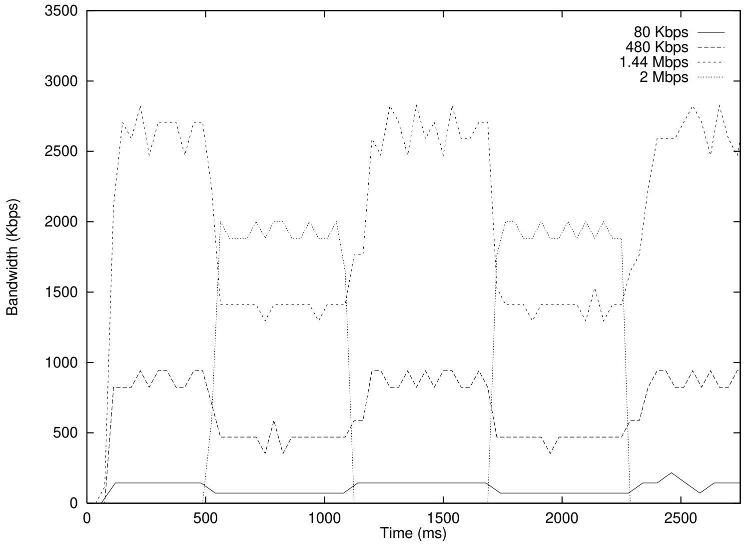
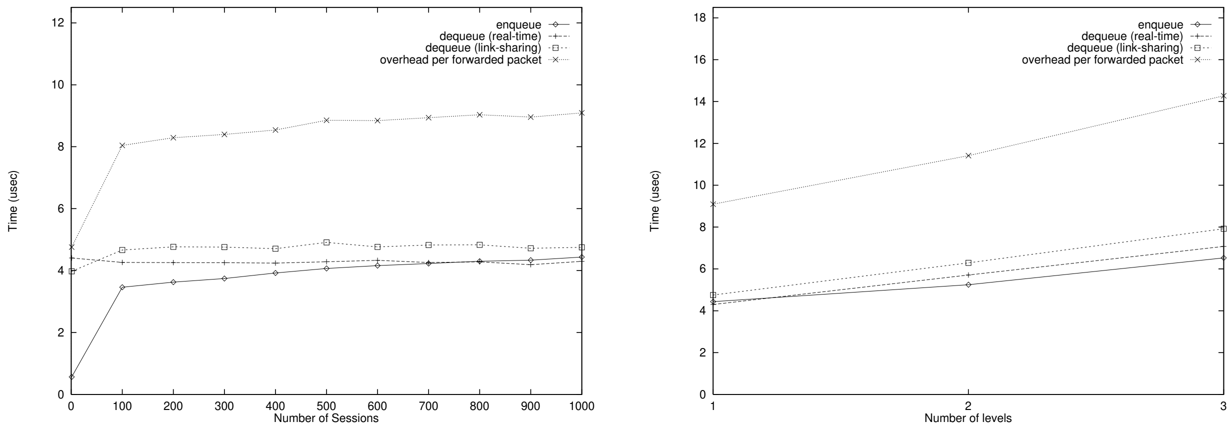


Figure 12: *Bandwidth distribution among four competing sessions.*



(a) (b)

Figure 13: *(a) The overheads for a flat hierarchy with 1, 100, $\cdots$, and 1000 sessions. (b) The overheads for a one-level, two-level, and three-level hierarchies, with each hierarchy having 1000 sessions.*

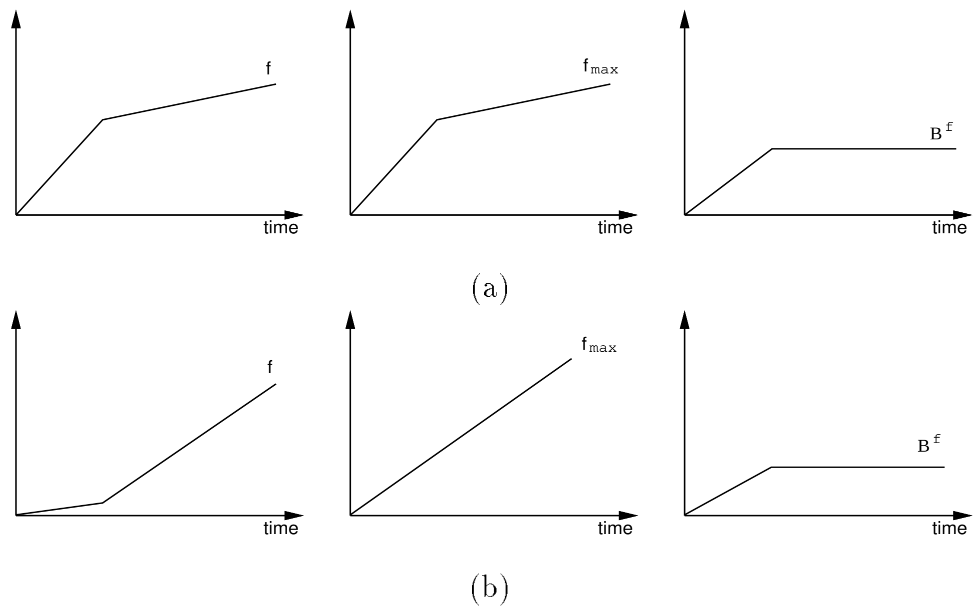


Figure 14: *The envelope ($f_{max}(\cdot)$) and the burstiness ($B^f(\cdot)$) functions associated to (a) a two-piece non-decreasing concave, and (b) a two-piece non-decreasing convex function, respectively.*