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

Ion Stoica, Hui Zhang, T. S. Eugene Ng
Carnegie Mellon University
Pittsburgh, PA 15213
e-mail: {istoica,hzhang,eugeneng}@cs.cmu.edu

Abstract— **In this paper, we study hierarchical resource management models and algorithms that support both link-sharing and guaranteed real-time services with priority (decoupled delay and bandwidth allocation). We extend the service curve based QoS model, which defines both delay and bandwidth requirements of a class in a hierarchy, 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, real-time and priority 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 service curves of leaf classes, thus ensures real-time and priority services, while minimizing the discrepancy between the actual services provided to and the services defined by the Fair Service Curve link-sharing model for the interior classes. We have implemented the H-FSC scheduler in NetBSD. By performing simulation and measurement experiments, we evaluate the link-sharing and real-time performances of H-FSC, and determine the computation overhead.**

I. INTRODUCTION

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 the real-time requirement, for each individual session, several recent works [1], [7], [15] have argued that it is also important to support hierarchical link-sharing service.

In hierarchical link-sharing, there is a class hierarchy associated with each link that specifies the resource allocation policy for the link. A class represents a traffic stream or 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, Carnegie Mellon University (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 its resource requirements, in this case, a bandwidth, which is the minimum amount of service that the traffic of the class should receive when there is enough demand.

There are several important goals that the hierarchical link-sharing service is aimed to achieve. First, each class should receive a certain minimum amount of resource if there is enough demand. In the example, CMU's traffic should receive at least 25 Mbps of bandwidth during a period when the aggregate traffic from CMU has a higher arrival rate. Similarly, if there is resource contention between traffic classes within CMU, the video traffic should get at least 10 Mbps. In the case where 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 a certain traffic class from CMU does not have enough traffic to fully utilize its minimum guaranteed bandwidth, other traffic classes from CMU have precedence to use this *excess* bandwidth over traffic classes from U. Pitt. While the above policy specifies that the CMU audio and video traffic classes have priority to use any 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. The 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 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, it is also important to support priority service in the sense that delay (both average delay and delay bound) and bandwidth allocation 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. Decoupling the delay and bandwidth 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 VII, they all suffer from important limitations. The fundamental problem is that with all three services, multiple requirements need to be satisfied simultaneously. In some cases this is impossible to achieve due to conflicting requirements. This problem is exacerbated by the fact that there is no formal definition of a hierarchical link-sharing service that specifies all these requirements.

In this paper, we consider an ideal model, called Fair Service Curve (FSC) link-sharing, that precisely defines 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)

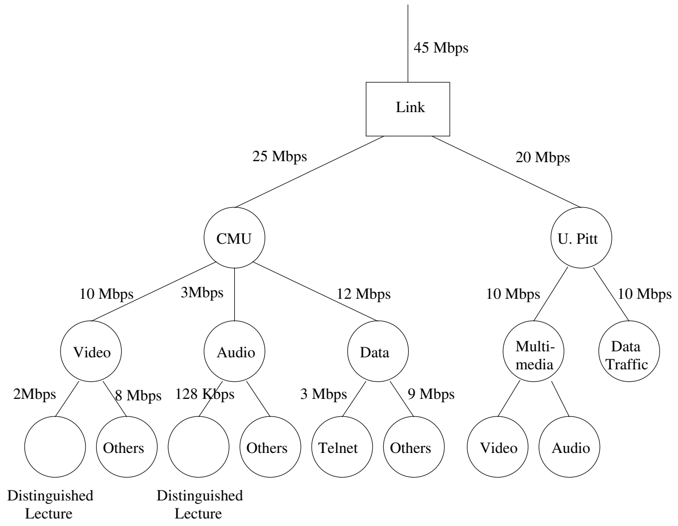


Fig. 1. *An Example of Link-Sharing Hierarchy.*

requirements. In this architecture, each class in the hierarchy is associated with a service curve. The ideal *Fair Service Curve link-sharing* model requires that (a) the service curves of all classes in the hierarchy are simultaneously guaranteed, and (b) the excess bandwidth unused by a class is distributed to its sibling classes fairly. Since the service curves of all classes are guaranteed simultaneously, the QoS of individual sessions (leaf classes in the hierarchy) and traffic aggregates (interior and possibly leaf classes in the hierarchy) are satisfied. In addition, delay and bandwidth allocation can be decoupled by choosing service curves of different shapes. Therefore, the Fair Service Curve link-sharing model gives a precise definition of a service that satisfies all the important goals of link-sharing, real-time, and priority services.

Unfortunately, as we will show in the paper, the ideal model cannot be realized at all time instances. In spite of this, the model serves two important purposes. First, unlike previous models, the new model explicitly defines the situations where 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 classes,
- minimize the short-term discrepancy between the amount of services provided to an interior class and the amount specified by the Fair Service Curve link-sharing model,
- allocate the excess bandwidth to sibling classes with bounded fairness

We have made the architecture level decision that whenever there is a conflict between performance goals, the performance guarantees of the leaf classes take precedence. We believe this is the right tradeoff as the performance of leaf classes is directly 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 in this framework.

The rest of the paper is organized as follows. Section II presents the Fair Service Curve link-sharing model and discusses the fundamental tradeoffs in approximating this model. Section III presents our solution, the Hierarchical Fair Service Curve (H-FSC) scheduler, followed by a discussion on its implementation complexity in Section IV. We analyze the delay and fairness properties of H-FSC in Section V, and evaluate its performance based on both simulation and measurement experiments in Section VI. We discuss related work in Section VII and conclude the paper in Section VIII.

II. 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 curve 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 this model.

A. Service Curve Based QoS Model

As discussed in Section I, we 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 $S_i(\cdot)$, where $S_i(\cdot)$ is a non-decreasing function, if for any time t_2 when session i is backlogged, there exists a time $t_1 < t_2$, which is the beginning of 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. A service curve is said to be *convex* if its second derivative is non-negative and is not the constant function zero, and it is said to be *concave* if its second derivative is non-positive and is not the constant function zero.

In the case in which the *server's* service curve is not concave, one algorithm that supports service curve guarantees is Service Curve Earliest Deadline first (SCED) [14]. With SCED, a deadline is computed for each packet using a per session deadline curve D_i and packets are transmitted in increasing order of their deadlines. The deadline curve D_i is computed such that in an idealized fluid system, session i's service curve is guaranteed if by any time t when session i is backlogged, at least $D_i(t)$ amount of service is provided to session i. Based on Eq. (1), it follows that

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

where $B_i(t)$ is the set of all time instances, no larger than t, when session i becomes backlogged, and $w_i(t_1) = w_i(0, t_1)$ is the total amount of service session i has received by time t_1. This gives the following iterative algorithm to compute D_i. When session i becomes backlogged for the first time, D_i is initialized to i's service curve $S_i(\cdot)$. Subsequently, whenever session i becomes backlogged again at time a_i^k (the beginning of session i's k-th backlogged period) after an idling period, D_i is updated according to the following:

$$D_i(a_i^k;t) = \min(D_i(a_i^{k-1};t), S_i(t-a_i^k) + w_i(a_i^k)), \quad t \geq a_i^k. \tag{3}$$

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

$$d_i = D_i^{-1}(a_i^k; w_i(t) + l_i). \tag{4}$$

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 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 delay and bandwidth allocation are taken into account in an *integrated* fashion, yet the allocation policies for these two resources are decoupled. This increases 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 a 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 D_i are the same as their service curves $S_i(\cdot)$. 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 $S_i(t)$ equals $A_i(u)$. 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-wise 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 delay for video packets does not come for free, that is, the delay for FTP packets increases consequently. However, this is acceptable since throughput rather than per packet delay is important to the FTP session.

While in theory any non-decreasing function can be used as a service curve, in practice only linear or piece-wise linear functions are used for simplicity. In general, a concave service curve results in a lower average and worst case delay for a session than a linear or convex service curve with the same guaranteed asymptotic rate. However, it is impossible to have concave service curves for all sessions and still reach high average utilization. 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 by time t. That is, the sum of the service curves over all sessions should be no more than the server's service curve.

B. Service Curve and Fairness

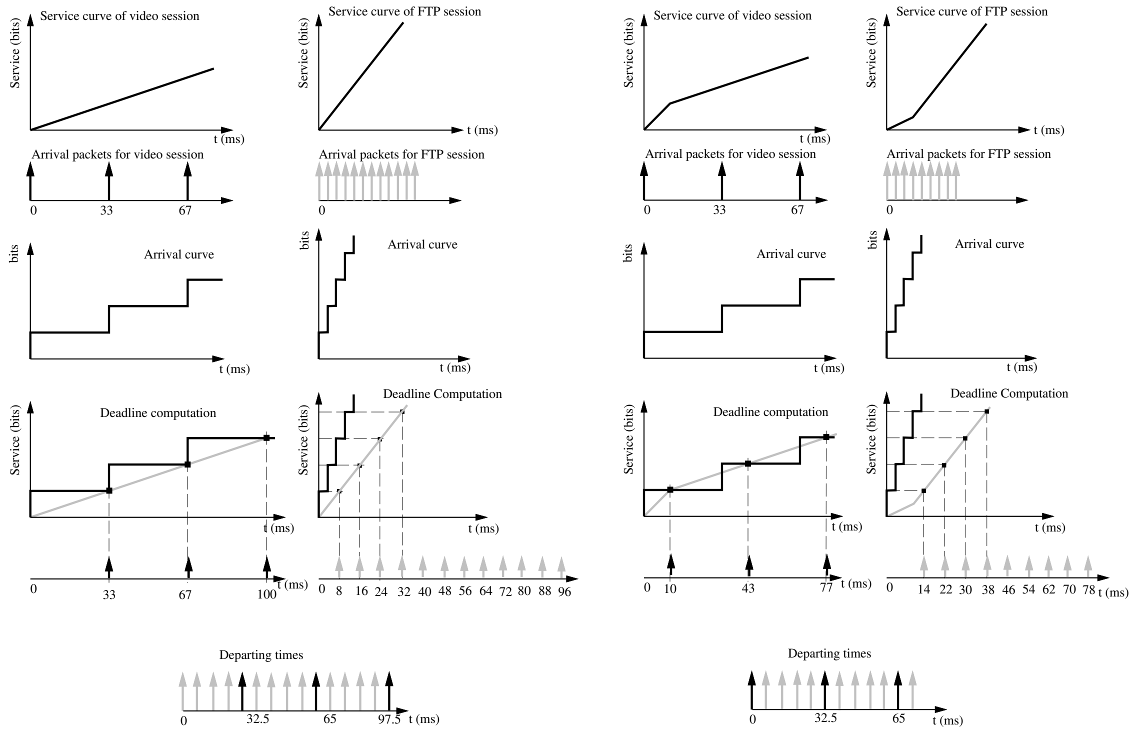
While the service curve is very general in specifying the minimum amount of service (in terms of bandwidth and delay) guaranteed to a session or a class, it does not specify how the *excess* service, which is the extra capacity of the server beyond what is needed to guarantee the service curves of all 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 arbitrarily long period because it received 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 fairness properties have been extensively studied for scheduling algorithms that only use sessions' rates as parameters, and there are several formal definitions of fairness, such as the relative fairness given by Golestani [9] 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 properties and argue that fairness is important even for scheduling algorithms that provide 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) the policy of distributing excess service to each of the currently active sessions, and (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 distributes the excess service to all backlogged sessions proportional to their minimum guaranteed rates. In addi-

[1]In theory, Fair Queueing and its corresponding fluid algorithm GPS can support more general service curves than linear curves [12], [19]. However, in practice, such a resource assignment has a number of limitations. See Section VII for a detailed discussion.



(a) (b)

Fig. 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.*

tion, it does not punish a 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 the 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. In this paper, we simply distribute the 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, the main motivation of link-sharing service is the *dynamic* sharing of resources among applications within one ancestor class. Such dynamic resource sharing is only meaningful if some applications in the class are *adaptive* – that is, during certain periods, they are able to send more than their minimum guaranteed bandwidth. We believe that taking advantage of the excess service in the context of hierarchical sharing is a part of the link-sharing service, and the applications should not be punished. Furthermore, even in a network that supports guarantees, it is still desirable to let applications to statistically share the fraction of resources that are either not reserved and/or not currently being used. We believe, when coupled with the right pricing model, a fair scheduler leads to higher application performance and lower call blocking rate as it encourages flexible applications to reserve less resources. For example, a video application may choose to make reservation only for its minimal transmission quality and use the excess service to increase its quality. In a system which penalizes a session for using excess service, such an adaptive application runs the risk of not receiving its minimum bandwidth if it uses excess service. As a result the application may simply choose to reserve more resources, rather than *always* transmitting at its minimal quality. Second, 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 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's 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;

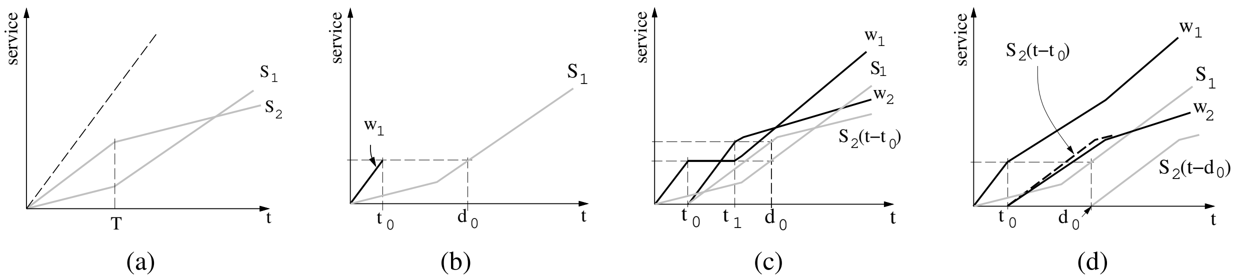


Fig. 3. *An example illustrating the "punishment" of a session under SCED: (a) the sessions' service curves. (b) session 1 is the only active session during* $(0, t_0]$. *(c) session 1 does not receive any service during* $(t_0, t_1]$, *after session 2 becomes active at* t_0. *(d) A modified version of SCED that tries not to penalize session 1 at all, but violates session* 2*'s service curve.*

and $2\beta > 1$, i.e., it is not possible to guarantee the peak rates of both sessions simultaneously.

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 ($t \geq t_s^i$) 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 $0 < t \leq t_0$ (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 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 the 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 $S_2^{-1}(w_2(t_1)) + t_0 \geq S_1^{-1}(t_0)$.

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 scheduler. 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 straight lines passing through the origin, SCED reduces to the well-known Virtual Clock discipline. While Virtual Clock is unfair [12], [20], 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. A PFQ 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 transformation 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. While we will describe the detailed algorithm in Section III, we use the example in Figure 3(d) to illustrate the concept. The main modification to SCED would be to use $S_2(t - d_0)$ instead of $S_2(t - t_0)$ in computing the packets' deadlines for session 2. 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 behaves identically to 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 shall see in the next section, in general this is impossible.

C. *Fair Service Curve Link-Sharing Model*

As discussed in the beginning of the paper, the important goals of hierarchical link-sharing are: to provide guaranteed QoS for each class, to allow priority (decoupled delay and bandwidth allocation) among classes, and to properly distribute 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 the fairness property for the purpose of excess bandwidth distribution, it is natural to use service curves to define the performance goals of link-sharing, real-time and priority services. In the Fair Service Curve link-sharing model there is a service curve associated with *each* class in the link-sharing hierarchy. The goal is then to (1) satisfy the service curves of all classes simultaneously, and (2) distribute the excess service fairly as defined in Section II-B. 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 class in the hierarchy, and its performance will be automatically guaranteed

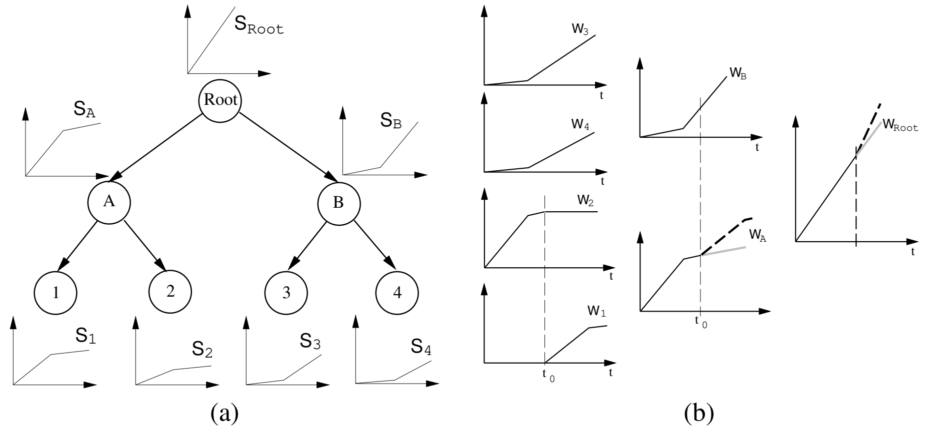


Fig. 4. *An example illustrating why it is not possible to guarantee the service curves of all the classes in the hierarchy. (a) The hierarchy and the service curves of each class. (b) The service received by each session when sessions 2, 3, and 4 become active at time* 0*; session 1 becomes active at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

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 curves of all classes, or (b) it is not possible to simultaneously guarantee both the service curves and satisfy the 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 needs 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 interval. Since in the context of service-curve-based schedulers, decoupling delay and bandwidth allocation is equivalent to specifying a non-linear service curve, this result translates into a fundamental conflict between link-sharing and real-time service when the delay and bandwidth allocation is decoupled.

To see why (b) is true, 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 $S_2(\cdot)$.

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 as 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 at all times while minimizing the discrepancy between the actual service allocated to each interior class and its fair service according to the model.

III. 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.

A. *Overview of the Algorithm*

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

In H-FSC, each leaf class i maintains a triplet (e_i, d_i, v_i), while each interior class i maintains only the parameter v_i. e_i and d_i represent the eligible time and the deadline associated with the packet at the head of class i's queue, and v_i is the virtual time associated with class i. The deadlines are assigned such that if the deadlines of all packets of a session are met, its service curve is guaranteed. The eligible 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. Eligible times signal when there is a potential conflict between link-sharing and real-time goals. When there are eligible packets in the system, there is a non-zero probability that the deadline of at least one packet is violated if the link-sharing instead of the real-time criterion is used. Since the real-time goal is more important, whenever there are eligible packets, the algorithm uses the real-time criterion to select among all eligible packets the one with the smallest

```
receive_packet(i, p) /* session i has received packet p */
  enqueue(queue_i, p);
  if (not active(i)) /* i was passive */
    update_ed(i, null, p); /* update E_i, D_i, compute e_i, d_i */
    update_v(i, p); /* update V_i, its ancestors; compute v_i */
    set_active(i); /* mark i and its ancestors active */

get_packet() /* get next packet to send */
  if (not active(root)) return;
  /* select by real-time criterion */
  i = min_{d_j} {j | leaf(j) ∧ active(j) ∧ (e_j ≤ current_time)};
  if (exists(i))
    p = dequeue(queue_i);
    update_v(i, p); /* update virtual time */
    if (not empty (queue_i))
      update_ed(i, p, head(queue_i));
    else
      set_passive(i); /* mark i and its ancestors passive */
  else /* select active session by link-sharing criterion */
    i = root;
    while (not empty(ActiveChildren(i)))
      i = min_{v_j} {j ∈ ActiveChildren(i)};
    p = dequeue(queue_i);
    update_v(i, p)
    if (not empty(queue_i))
      update_d(i, p, head(queue_i)) /* update d_i only */
    else
      set_passive(i); /* mark i and its ancestors passive */
  send_packet(p);
```

Fig. 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.*

deadline. At any given time when there is no eligible packet, the algorithm applies the link-sharing criterion recursively, starting from the root and stopping at a leaf class, selecting at each level the class with the smallest virtual time. While deadline and eligible 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 perfectly fair system, the virtual times of all active sibling classes at each level in the hierarchy should be identical. The objective of the link-sharing criterion is then to minimize the discrepancies between virtual times of sibling classes. The pseudo code of H-FSC is given in Figure 5. A leaf class is said to be *active* if it has at least one packet enqueued. An interior class is *active* if at least one of the leaf classes among its descendents is active. Otherwise a class is said to be *passive*. In computing the eligible time, the deadline, and the virtual time, the algorithm uses three curves, one for each parameter: the eligible curve E_i, the deadline curve D_i , and the virtual curve V_i. The exact algorithms to update these curves are presented in Sections III-B and III-C.

There are several noteworthy points about this algorithm. First, while H-FSC needs to use two packet selection criteria to support link-sharing and real-time services, Hierarchical Packet Fair Queueing (H-PFQ) [1] selects packets solely based on the link-sharing criterion, 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 priority, i.e., decoupled delay and bandwidth allocation, by guaranteeing non-linear service curves. As we have shown in Section II, it is in general 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 eligible time to determine which criterion to use. Second, the algorithm uses three types of time parameter: deadline, eligible time, and virtual time. While leaf classes maintain all three parameters, the interior classes maintain only the virtual time parameter. This is because deadlines and eligible times are used for the purpose of guaranteeing the service curves, and H-FSC provides service curve guarantees 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. Note that although in H-FSC virtual times are computed based on the classes' service curves to achieve fairness and hierarchical link-sharing, H-FSC can potentially use other policies to distribute the excess service. We choose to use the same curve for both the real-time and link-sharing policies for its simplicity. The same tradeoff was made by many of the previous fair queueing algorithms [2], [6], [9], [11], [13]. A third point to notice is that while all three parameters are time values, they are measured with respect to different clocks. Deadlines and eligible times are measured in wall-clcok time. In contrast, the virtual time of a class is measured with respect to the total amount of service received by the class and so only the relative differences between virtual times of sibling classes are important.

Finally, 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 the link-sharing criterion, 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, in H-FSC, the delay bound of a leaf class is determined by the real-time criterion, which considers only the leaf classes. Therefore, the delay bound is independent of the class hierarchy.

B. Deadline and Eligible Time

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

For each leaf class i, the algorithm maintains two curves, one for each time parameter: the eligible curve $E_i(a_i^k;\cdot)$ and the deadline curve $D_i(a_i^k;\cdot)$, where a_i^k represents the beginning of the k-th active (backlogged) period of class i. 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 criterion*. Thus c_i represents the total amount of service that the class has received when selected under the real-time criterion. Like SCED, the deadline curve $D_i(a_i^k;\cdot)$ is initialized to its service

update_ed$(i, p, next\_p)$
  static a;
  if (**not active**(i))
    $/*$ *session i becomes active* $*/$
    $a = current\_time;$
    update_DC(i, a); $/*$ *update deadline curve* D_i *(Eq. (7))* $*/$
    update_EC(i, a); $/*$ *update eligible curve* E_i *(Eq. (11))* $*/$
  if $(p \, != null)$
    $c_i = c_i + \mathbf{length}(p);$
  $/*$ *update deadline (Eq. (4))* $*/$
  $d_i = D_i^{-1}(a; c_i + \mathbf{length}(next\_p));$
  $e_i = E_i^{-1}(a; c_i)$; $/*$ *update eligible time* $*/$

(a)

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

(b)

Fig. 6. *(a) The function which updates the deadline and the eligible curves, and computes the deadline and the eligible time for each leaf class (session). Note that the eligible and the deadline curves are updated only when the session becomes active. (b) The function which updates the deadline, when the session has been served by the link-sharing criterion. This is because the new packet at the head of the queue may have a different length.*

curve $S_i(\cdot)$, and is updated each time session i becomes active at time a_i^k according to

$$D_i(a_i^k; t) = \min(D_i(a_i^{k-1}; t), S_i(t - a_i^k) + c_i(a_i^k)), \quad t \geq a_i^k. \tag{7}$$

Here we use $c_i(a_i^k)$ to denote the total service[2] received by class i by the real-time criterion at time a_i^k. Since c_i does *not* change when a session receives service via the link-sharing criterion, the deadlines of future packets are not affected (see Figure 6). This is the essence of the "non-punishment" aspect of the fairness property.

While deadlines are used to guarantee service curves of leaf classes, eligible times are used to arbitrate which one of the two scheduling criteria is used 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 a_i^k, 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's rate, it is impossible to satisfy the service curves of all sessions.

The only solution is to have the server allocates "enough" service in advance to active sessions by the real-time criterion, 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 criterion despite another packet having a smaller virtual time, there is a departure from the ideal link-sharing distribution. Therefore, to minimize the deviation from the ideal FSC link-sharing model, we want to serve packets using the real-time criterion *only* when there is a danger of violating the guarantees for leaf classes in the future.

In H-FSC, eligible times are used to arbitrate which one of the two scheduling 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 amount of 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 $R \times (t' - t)$, where R is the server capacity. 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 [18]. Because the above condition holds for any future time t', we have

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

where a_i represents the last time, no larger than t, when session i became active, $\mathcal{A}(t)$ denotes the set of active sessions at time t, $\mathcal{P}(t)$ denotes the set of passive sessions at time t, and $[x]^+$ denotes $\max(x, 0)$. The above equation reads as follows. In the worst case, all active sessions continue to remain active up to time t', and all passive sessions become immediately active at time t and remain active during the entire interval $(t, t']$. As a result, the maximum amount of service required over the interval $(t, t']$ by the sessions that are already active at time t is $\sum_{i \in \mathcal{A}(t)} (D_i(a_i; t') - D_i(a_i; t))$, while the maximum amount of service required by the sessions that are passive up to time t over the same interval is $\sum_{i \in \mathcal{P}(t)} (D_i(t; t') - D_i(t; t))$. Since all sessions together can receive at most $R \times (t' - t)$ of service during the interval $(t, t']$, and since by time t the active sessions should have received at least $\sum_{i \in \mathcal{A}(t)} D_i(a_i; t)$ in order to satisfy their service curves, the above equation follows.

Thus, $E(t)$ represents the minimum amount of service that should be allocated to the active sessions by time t using the real-time criterion in order to guarantee the service curves of all sessions in the future. The remaining (excess) service can be allocated by the link-sharing criterion. Furthermore, 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 $E(t)$ of service to the active sessions by time t. A possible algorithm would then be simply to allocate $E(t)$ of service to active sessions by the real-time criterion, and redistribute the excess service according to the link-sharing criterion. 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

[2] Note that Eq. (7) is the same as Eq. (3), except that c_i is used instead of w_i.

the passive ones. Since according to Eq. (7), the deadline curve of a session depends on the time when the 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 if session i becomes active at time t, we have (see Eq. (7)):

$$D_i(t;t') - D_i(t;t) \leq S_i(t'-t), \ \ t' \geq t. \quad (9)$$

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

$$\begin{aligned}
E(t) = & \sum_{i \in \mathcal{A}(t)} D_i(a_i;t) \\
& + [\max_{t'>t}(\sum_{i \in \mathcal{A}(t)} (D_i(a_i;t') - D_i(a_i;t)) \quad (10) \\
& + \sum_{i \in \mathcal{P}(t)} (D_i(t;t') - D_i(t;t)) - R \times (t'-t))]^+ \\
\leq & \sum_{i \in \mathcal{A}(t)} D_i(a_i;t) + [\max_{t'>t}(\sum_{i \in \mathcal{A}(t)} (D_i(a_i;t') - D_i(a_i;t)) \\
& + \sum_{i \in \mathcal{P}(t)} S_i(t'-t) - R \times (t'-t))]^+ \\
\leq & \sum_{i \in \mathcal{A}(t)} D_i(a_i;t) + [\max_{t'>t}(\sum_{i \in \mathcal{A}(t)} (D_i(a_i;t') - D_i(a_i;t)) \\
& + \sum_{i \in \mathcal{P}(t)} S_i(t'-t) - \sum_{i \in \mathcal{A}(t) \cup \mathcal{P}(t)} S_i(t'-t))]^+ \\
= & \sum_{i \in \mathcal{A}(t)} D_i(a_i;t) + [\max_{t'>t}(\sum_{i \in \mathcal{A}(t)} (D_i(a_i;t') - D_i(a_i;t) \\
& - S_i(t'-t)))]^+ \\
\leq & \sum_{i \in \mathcal{A}(t)} (D_i(a_i;t) + [\max_{t'>t}(D_i(a_i;t') - D_i(a_i;t) \\
& - S_i(t'-t))]^+).
\end{aligned}$$

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

$$\begin{aligned}
E_i(a_i;t) = D_i(a_i;t) + [\max_{t'>t}(D_i(a_i;t') - D_i(a_i;t) \\
- S_i(t'-t))]^+, \ \ t \geq a_i, \quad (11)
\end{aligned}$$

where again a_i represents the time when session i becomes active. The eligible curve $E_i(a_i;t)$ determines the maximum amount of service received by session i at time t by the real-time criterion, when session i is continuously backlogged during $(a_i,t]$. Since $\sum_{i \in \mathcal{A}(t)} E_i(a_i;t) \geq E(t)$, we have a sufficient condition. $E_i(\cdot;\cdot)$ is updated every time session i becomes active by the function **update_EC** according to Eq. (11) (see Figure 6). It is important to note that even though the formula, which is applicable to algorithms with service curves of arbitrary shapes looks complicated the eligible curves are actually quite simple to compute in the specific cases that we are interested in. For example, for a session with a concave service curve the eligible curve is the same as the deadline curve. Intuitively this is easy to understand as the minimum service rate required by a session with a concave service curve will not increase in the future, thus there is no need to provide future service for it. Similarly, for a session with a two-piece linear convex service curve (first slope α, second slope β, where $\beta > \alpha$), the eligible curve is the linear curve with the slope of β.

```
update_v(i, p)
  static a;
  n = parent(i);
  if (not active(i))
    a = current_time; /*class i becomes active */
    v_i = max(v_i, v_n^s); /* v_n^s = (min_{i∈ActiveChildren(n)}(v_i)+
max_{i∈ActiveChildren(n)}(v_i))/2 */
    update_VC(i); /* update eligible curve V_i by Eq. (12) */
    if (active(n))
      return;
  else /*class i is already active */
    w_i = w_i + length(p);
    v_i = V_i^{-1}(a; w_i);
  if (n ≠ ROOT)
    update_v(n, p);
```

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

C. 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 use a generalized version of virtual time to achieve hierarchical link-sharing.

Each Fair Queueing algorithm maintains a system virtual time $v^s(\cdot)$ [9]. 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 that session i has received by time t, and $f_i(t)$ represents the sum between $s_i(t)$ and the normalized service that session i should receive when the packet at the head of its queue is served. 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 $v^s(t)$. In a H-PFQ system, each class keeps a virtual time function and the goal is to minimize the discrepancies among the virtual times of all sibling classes 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 virtual start time of the packet currently being served [11], the virtual finish time of the packet currently being served [9], and the minimum between the current value of a linear function that advances at the server's rate, and the smallest of the virtual start times of all packets at the heads of currently backlogged queues [1]. Examples of packet selection policies are Smallest Start time First (SSF) [11], Smallest Finish time First (SFF) [9], and Smallest Eligible Finish time First (SEFF) [2], [17]. The choice of different system virtual time functions and packet selection policies affects the real-time and fairness properties of the resulting 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 that class i has received by time t. In H-FSC, virtual times are used by the link-sharing criterion to distribute service among the hierarchy according to classes' service curves. The link-sharing criterion is used to select the next packet only when the real-time criterion is not used. Since the real-time guarantees for leaf classes are ensured by the real-time criterion, the effect on performance by having different system virtual time functions and packet selection algorithms in the link-sharing criterion is less critical. In H-FSC we use the SSF policy and the 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 the maximum virtual start times among the active children of class i. By doing so, we ensure that the discrepancy between the virtual times of any two active sibling leaf classes is bounded (see Section V). It is interesting to note that setting v_i^s to either $v_{i,min}$ or $v_{i,max}$ results in a discrepancy proportional to the number of sibling classes.

In H-FSC, $v_i(t)$ is iteratively computed by using the previous virtual time function and the class' service curve. Virtual times are updated when a packet starts being serviced or a class becomes active. The function **update_v** for this purpose is shown in Figure 7. Notice that **update_v** recursively updates the virtual times and the virtual curves in the hierarchy by following child-parent links till it reaches the root or a parent class that is already active.

In the algorithm, we actually maintain a virtual curve V_i, the inverse function of v_i, instead of v_i. When class i becomes active for the first time, V_i is initialized to i's service curve $S_i(\cdot)$. V_i is then updated by using the **update_VC** function every time the class becomes active at time a_i^k, the beginning of the k-th active period, based on the following formula

$$V_i(a_i^k; v) = \min(V_i(a_i^{k-1}; v), S_i(v - v_{p(i)}^s(a_i^k)) + w_i(a_i^k)), \quad v \geq v_{p(i)}^s(a_i^k), \quad (12)$$

where $w_i(a_i^k)$ is the total amount of service received by class i by time a_i^k, both by the link-sharing and the real-time criteria, while $v_{p(i)}^s(a_i^k)$ is the system virtual time associated to the parent of class i. Note that we use v instead of t in the above equation to reflect the fact that V_i is a function of the virtual time. Finally, it is worth noting that when $S_i(\cdot)$ is a straight line with slope r_i, from Eq. (12) we have $V_i(a_i^k; v) = r_i v$. Then, the virtual time v_i is simply $V^{-1}(a_i^k; w_i) = w_i/r_i$, which is exactly the virtual time of session i in PFQ algorithms.

IV. Implementation Issues and Complexity

The functions **receive_packet** and **get_packet** described in Figure 5 are called each time a packet arrives or departs. In our current implementation we maintain two requests per session, one characterized by the eligible time and deadline, called the *real-time request*, and the other characterized by the virtual time, called the *link-sharing request*. For maintaining the real-time requests we can use either an augmented binary tree data structure as the one described in [16], 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 it 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 on average. The link-sharing requests are stored in a heap based on their virtual times.

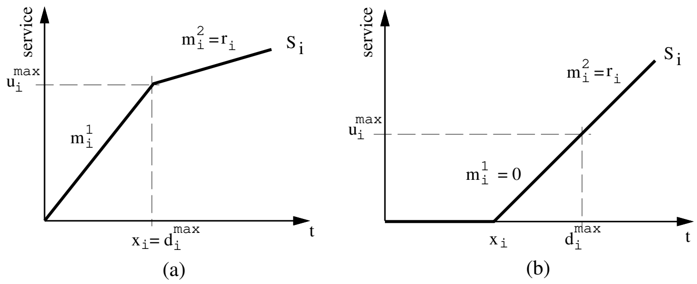


Fig. 8. *The service curve associated with a session i is characterized by its maximum delay d_i^{max}, maximum unit of work u_i^{max}, and average rate r_i. (a) If $u_i^{max}/d_i^{max} > r_i$, the service curve is concave; (b) otherwise, it is convex.*

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 the complexity can be significantly reduced by considering only piece-wise linear curves.

Up to this point, our results apply to classes with general non-decreasing service curves. However, for simplicity, in our implementation we consider concave and convex service curves only. Each session i is characterized by three parameters: the largest unit of work, denoted u_i^{max}, for which the session requires delay guarantee, the guaranteed delay d_i^{max}, and the session's average rate r_i. As an example, if a session requires per packet delay guarantee, then u_i^{max} represents the maximum size of a packet. Similarly, a video or an audio session can require per frame delay guarantee, by setting u_i^{max} to the maximum size of a frame. The session's requirements are mapped onto 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 update operation. It is worth noting that although all two-piece linear *concave* curves exhibit this nice property, this is not true for all two-piece linear convex curves. In fact, it can be shown that only two-piece linear convex service curves which have their first segment parallel to the x-axis have this property. 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. For this purpose we use the y-coordinate of the intersection between the two segments and

update_DC(i, a)
  if $((m_i^1 > m_i^2)$ **and** $(c_i + y_i^S - y_i > m_i^2 \times (a + x_i^S - x_i)))$
    $/*$ *D_i concave and intersects $S_i(t-a) + c_i$* $*/$
    $temp = y_i - m_i^2 x_i$; $/*$ *compute intersection point* $*/$
    $x_i = (c_i - m_i^1 a - temp)/(m_i^2 - m_i^1)$;
    $y_i = m_i^2 x_i + temp$;
  else
    $x_i = a + x_i^S$;
    $y_i = c_i + y_i^S$;

Fig. 9. *The function which updates the deadline curve D_i. a respresents the time when the session becomes active, c_i is the amount of service that has been received by session i by the real-time criterion, x_i and y_i are the coordinates of the inflexion point of the deadline curve, while x_i^S and y_i^S are the coordinates of the inflexion point of $S_i(\cdot)$.*

denote it y_i. The pseudocode for updating a deadline curve is presented in Figure 9. The only parameters that are modified are the coordinates of the segments intersection point x_i and y_i; the slopes of the two segments, m_i^1 and m_i^2, remain unchanged. The deadline curve, as well as the virtual and eligible curves, is updated **only** when the state of a 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 performed by **update_VC**. Since this function is very similar to **update_DC** — the only difference is that instead of using c_i and a, we use the total service w_i and the virtual time $v^s_{p(i)}$, respectively — we do not show it here.

Although from Eq. (11) it appears that the computation of the eligible curve is quite complex, it turns out that it can be done very efficiently in our case: 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 is simply a line that starts at the same point as the first segment of the deadline curve, and has the same slope as the deadline curve's 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 $O(\log n)$ to execute per packet arrival or packet departure, which is similar to other packet scheduling algorithms [1].

V. 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 [18]. For the rest of the discussion, we consider the *arrival* time of a packet to be the time when the last bit of the packet has been received, and the *departing* time to be the time when the last bit of the packet has been transmitted.

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

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

The next theorem gives a tight delay bound for H-FSC. In conjunction with the previous theorem, this result shows that, in H-FSC, the service curves are guaranteed to 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.

Unlike H-PFQ, the delay bound of H-FSC does *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 H-FSC for leaf classes, by giving a bound on the discrepancy between the actual service distribution and the ideal link-sharing model.

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

From the theorem, the following corollary immediately follows:

Corollary 1 *In H-FSC, for any two sibling leaf classes i and j that are continuously active during a time interval $(t_1, t_2]$, the following holds,*

$$|\ (v_i(t_2) - v_i(t_1)) - (v_j(t_2) - v_j(t_1))\ | < B, \tag{13}$$

where B depends on the characteristics of the service curves of sessions i and j.

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 [9].

Unlike the discrepancy between two sibling leaf classes which is bounded by a value that depends on service curves associated with classes i and j only, in the case of two interior sibling classes, this discrepancy depends on *all* sessions in the system. This is because the scheduler uses the real-time criterion whenever a session is eligible, independent of the position of the session in the hierarchy. Thus, the bound for the discrepancy between two interior classes increases with the number of sessions in the system. To reduce this discrepancy, a possible solution is to use the global eligible curve E, computed by Eq. (8), instead of the individual sessions' eligible curves. However, as discussed in Section III-B, this increases the complexity of H-FSC. How much we can reduce the discrepancy and how to reduce the complexity of computing E are topics of future research.

VI. 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 simulator and the NetBSD implementation share basically the same 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 eligible time and deadline manipulations. In NetBSD, besides the scheduler, we have also

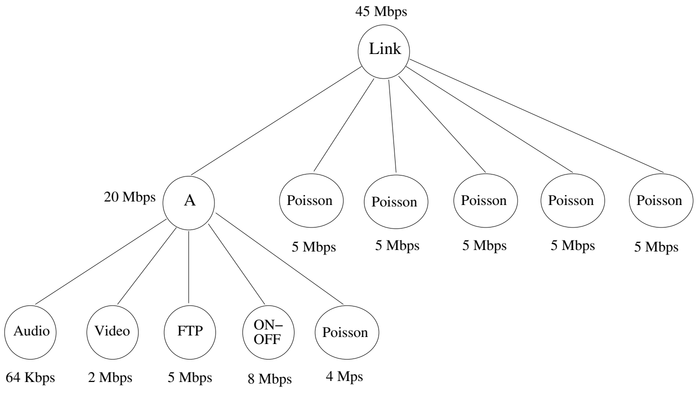


Fig. 10. *Class Hierarchy.*

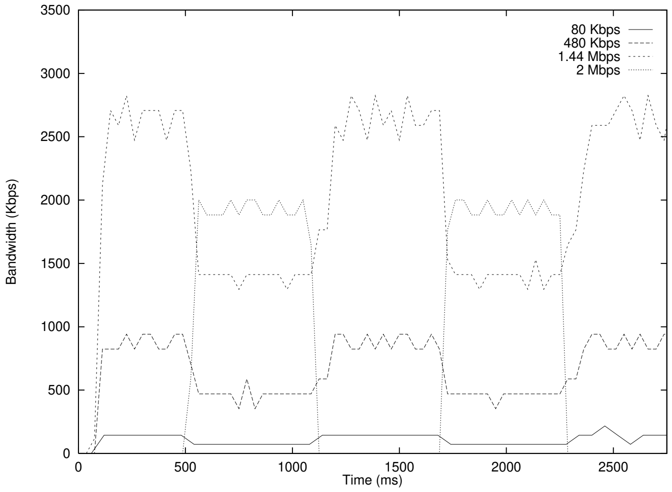


Fig. 12. *Bandwidth distribution among four competing sessions.*

implemented a packet classifier that maps IPv4 packets to the appropriate classes in the hierarchy.[3]

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 this 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.

A. 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 continuously 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 $S_{FTP} = (u_{FTP}^{max} = 4\text{ KB}, d_{FTP}^{max} = 16.25\text{ ms}, r_{FTP} = 5\text{ Mbps})$. The service curves of all the other sessions and classes are linear.

[3]This implementation is now publicly available for both NetBSD and FreeBSD at `http://www.cs.cmu.edu/~hzhang/HFSC/`.

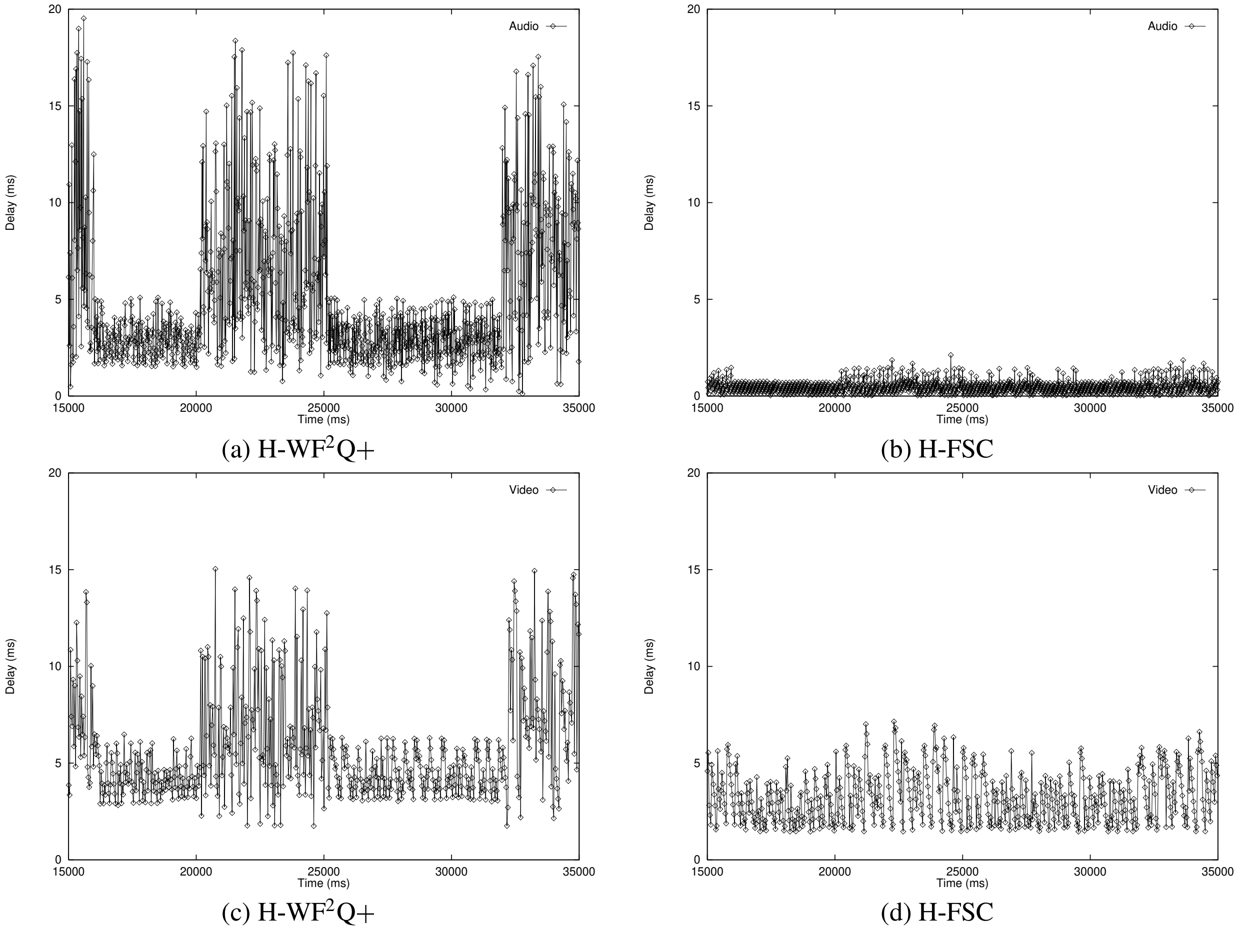
Figure 11 shows the delay distribution for the audio and video sessions under H-WF^{2}Q+ and H-FSC. Clearly, H-FSC achieves much lower delays for both audio and video sessions. The reduction in delay with H-FSC is especially significant for the audio session. This is a direct consequence of H-FSC's ability 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.[4] 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.

B. 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

[4]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 within a time interval also increases by approximately 40 %.



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

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

Fig. 11. *Delay for audio and video sessions.*

four sessions at 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.

C. 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 essentially 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 selecting a packet 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 very little as the number of sessions increases from 100 to 1000. This is to be 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

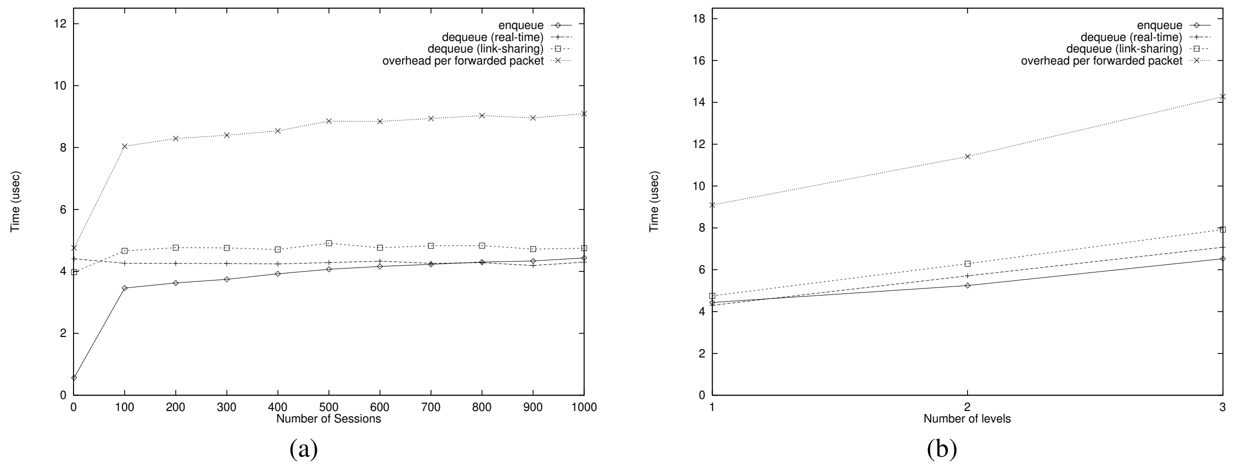


Fig. 13. *(a) The overheads for a flat hierarchy with 1, 100, $\cdots$, and 1000 sessions. (b) The overheads for a one-level, two-level, and three-level hierarchies, with each hierarchy having 1000 sessions.*

forward over 83,000 packets per second.[5]

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 $0.3\ \mu$s) 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 $8.79\ \mu$s, while for the three-level hierarchy it is $11.54\ \mu$s.

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

VII. Related Work

Class Based Queueing [7] and Hierarchical Packet Fair Queueing [1] are two algorithms that aim to support hierarchical link-sharing, real-time and priority 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 serves eligible packets using a static priority policy.

The key difference between H-FSC and CBQ is that H-FSC is designed using a formal approach. 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 [8], [21], 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 [8], [21]. In addition, in order for the end-to-end delay bound of a session to not be affected by the regulating delay, the session's parameters need to be consistent among all regulators in the network. In CBQ, the regulation process is affected by the link-sharing structure and policy, which are independently set at each router. 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-

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

time) and interior 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 [7]. 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 in H-FSC, priorities for packets are dynamically assigned based on service curves, while in 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 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 they 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 of 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 specify 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 with 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 [12], [19], it has been shown that more general service curves other 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 are policed at the sources. Therefore, sources will not be able to opportunistically utilize the excess bandwidth available in the network by sending more traffic than reserved. It is unclear whether link-sharing can be supported in such a network. In H-FSC, the scheduler guarantees a minimum service curve to a session regardless of the behavior of other sessions in the network. In addition, it does not require that a 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 still achieve a larger schedulability region than GPS with general resource assignments.

Fair Airport (FA) Schedulers proposed in [10] combine a Rate Controlled Service Discipline with Start-time Fair Queueing (SFQ) [11]. The concept of using two scheduling disciplines, one to enforce the real-time criterion, and the other to enforce the link-sharing criterion, is similar to H-FSC. The key difference is that while in FA the link-sharing criterion considers only the excess service, in H-FSC the link-sharing criterion considers the *entire* service allocated to a class. At the algorithmic level this difference is reflected by the fact that in FA the virtual time of a session is not updated when a packet is served by the real-time criterion.

VIII. 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 of leaf classes while minimizing the discrepancy between the actual service allocated and the service it should be allocated by the ideal FSC link-sharing model to the interior classes. 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] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Journal of Internetworking Research and Experience*, pages 3–26, October 1990. Also in Proceedings of ACM SIGCOMM'89, pp 3-12.

[7] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995.

[8] 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.

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

[10] P. Goyal and H. M. Vin. Fair airport scheduling algorithms. In *Proceedings of NOSSDAV'97*, St. Louis, MI, May 1997.

[11] 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.

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

[13] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control - the single node case. In *Proceedings of the INFOCOM'92*, 1992.

[14] 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.

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

[16] 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.

[17] 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.

[18] I. Stoica, H. Zhang, and T.S. E. Ng. A hierarchical fair service curve algorithm for link-sharing, real-time and priority services. Technical Report CMU-CS-97-154, Carnegie Mellon University, July 1997.

[19] 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.

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

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