Hierarchical Packet Fair Queueing Algorithms

Jon C. R. Bennett
FORE Systems Inc.
jcrb@fore.com

Hui Zhang*
Carnegie Mellon University
hzhang@cs.cmu.edu

Abstract

In this paper, we propose to use the idealized Hierarchical Generalized Processor Sharing (H-GPS) model to simultaneously support guaranteed real-time, rate-adaptive best-effort, and controlled link-sharing services. We design Hierarchical Packet Fair Queueing (H-PFQ) algorithms to approximate H-GPS by using one-level variable-rate PFQ servers as basic building blocks. By computing the system virtual time and per packet virtual start/finish times in unit of bits instead of seconds, most of the PFQ algorithms in the literature can be properly defined as variable-rate servers. We develop techniques to analyze delay and fairness properties of variable-rate and hierarchical PFQ servers. We demonstrate that in order to provide tight delay bounds with an H-PFQ server, it is essential for the one-level PFQ servers to have small Worst-case Fair Indices (WFI). We propose a new PFQ algorithm called WF^2Q+ that is the first to have all the following three properties: (a) providing the tightest delay bound among all PFQ algorithms, (b) having the smallest WFI among all PFQ algorithms, and (c) having a relatively low asymptotic complexity of O(log N). Simulation results are presented to evaluate the delay and link-sharing properties of H-WF^2Q+ , H-WFQ, H-SFQ, and H-SCFQ.

1 Introduction

Future integrated services networks will support multiple service classes that include real-time service, best-effort service, and others. In addition, they will need to support link-sharing [6], which allows resource sharing among traffic streams that are grouped according to administrative affiliation, protocol, traffic type, or other criteria. Figure 1 (a) shows an example where there are 11 agencies sharing the output link. The administrative policy dictates that Agency A1 gets at least 50% of the link bandwidth whenever it has traffic. In addition, to avoid starvation of the best-effort traffic, of the 50% of the bandwidth assigned to A1, best-effort traffic should get at least 20% if there is sufficient demand.

*This research is sponsored by DARPA under contract numbers N66001-96-C-8528 and N00174-96-K-0002, and by a NSF CAREER AWARD under grant number NCR-9624979. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Government.

It is important to design mechanisms to meet the goals of link sharing and different service classes simultaneously. The fluid Hierarchical Generalized Processor Sharing (H-GPS) system provides a general and flexible framework to support hierarchical link sharing and traffic management for different service classes. H-GPS can be viewed as a hierarchical integration of one-level GPS servers. With a one-level GPS, there are multiple packet queues, each associated with a service share. During any time interval when there are backlogged queues the server services all backlogged queues simultaneously in proportion to their corresponding service shares. With an H-GPS server, the queue at each internal node is a logical one, and the service it receives is distributed instantaneously to its child nodes in proportion to their relative service shares. This service distribution follows the hierarchy until it reaches the leaf nodes where there are physical queues.

It has been shown that with a one-level GPS: (1) an end-to-end delay bound can be provided to a session if the traffic on that session is leaky bucket constrained [12], (2) bandwidth is fairly distributed to competing sessions [5], and (3) the sources can accurately estimate the available bandwidth to them in a distributed fashion [9]. The first property forms the basis for supporting real-time traffic [2] and the third property enables robust and distributed end-to-end traffic management algorithms for best-effort traffic [9, 14]. H-GPS will maintain the first and the third properties, but distribute excess bandwidth unused by a session according to the hierarchy rather than just service shares of sessions. Therefore, the simple H-GPS configuration in Figure 1 (b) simultaneously supports all three goals, namely, link-sharing, real-time traffic management, and best-effort traffic management.

While H-GPS provides a simple model for supporting integrated services networks, it is defined in a hypothetical fluid system that cannot be precisely implemented. In this paper, we design packet approximation algorithms of H-GPS. In the literature, a number of one-level Packet Fair Queueing (PFQ) algorithms have been proposed to approximate the fluid GPS algorithm [1, 5, 7, 8, 12, 13, 17]. To reduce the implementation complexity, they all use the notion of a system virtual time function that tracks the progress in the fluid system. As we will show in Section 2, the same technique based on a single system virtual time function does not apply to packet algorithms approximating H-GPS.

In this paper, we propose to approximate H-GPS by using one-level PFQ servers as basic building blocks and organizing them in a hierarchical structure. The resulted Hierarchical Packet Fair Queueing (H-PFQ) algorithms should have the following properties: (1) tight per session delay bounds that are comparable to a H-GPS server, (2) bandwidth distribution in a hierarchical fashion that is similar to a H-GPS server, and (3) a relatively low complexity. To

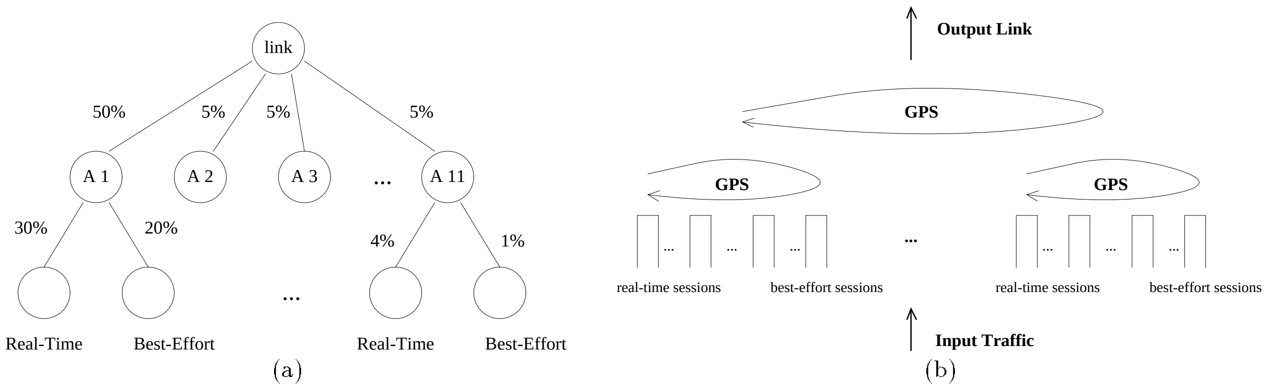


Figure 1: A Link Sharing Example

construct such a H-PFQ server, the one-level server needs to have the following properties: (a) tight per session delay bound as compared to the one-level GPS server, and (b) a relatively low complexity. In addition, as we will demonstrate, to achieve tight delay bounds in the H-GPS server, one-level PFQ servers also need to provide (c) small Worst-case Fair Indices (WFI's) as defined in [1]. Most of the previously proposed PFQ algorithms [5, 7, 8, 12, 17] do not have small WFI's. In fact, they all have WFI's that grow proportionally to the number of queues in the system. As a result, the delay bounds provided by H-PFQ servers made of these PFQ's are much larger than those provided by H-GPS. The only exception is Worst-case Fair Weighted Fair Queueing algorithm (WF^2Q), which is proven to provide smallest WFI's among all PFQ algorithms [1]. However, WF^2Q uses a system virtual time function with a complexity of O(N).

We propose a new algorithm that maintains all the important properties of WF^2Q, but has a lower complexity than WF^2Q. We call the new algorithm WF^2Q+ . Simulation results are presented to illustrate the advantages of H-WF^2Q+ over H-WFQ, H-SFQ, and H-SCFQ.

2 Fluid and Packet Systems

Throughout the paper, we discuss two types of systems: fluid system in which the traffic is infinitely divisible and multiple traffic streams can receive service simultaneously, and packet systems in which only one traffic stream can receive service at a time and the minimum service unit is a packet. While fluid systems cannot be realized in the real world, they are conceptually simple and some of them have properties that are highly desirable for network control. For these fluid systems, people have studied the corresponding packet approximation algorithms.

In this section, we first review Generalized Processor Sharing (GPS), and illustrate how it can be approximated by packet algorithms based on virtual time functions. We then define Hierarchical GPS (H-GPS) and show that the same technique cannot be applied directly to H-GPS.

2.1 Packet Approximation of GPS

A one-level GPS server with N queues is characterized by N positive real numbers, $\phi_1, \phi_2, \cdots, \phi_N$. Let $W_i(t_1, t_2)$ be the amount of session i traffic served in the interval $[t_1, t_2]$, $W(t_1, t_2)$ be the total amount of service provided by the server during the same period. A work-conserving GPS server is defined as one for which

$$\frac{W_i(t_1,t_2)}{\phi_i} = \frac{W(t_1,t_2)}{\sum_{j\in B(t_1)} \phi_j} \quad \forall i \in B(t_1) \tag{1}$$

holds for any interval $[t_1, t_2)$ during which $B(\tau)$, the set of backlogged sessions at time τ, does not change.

There are two noteworthy points. First, in the definition of the GPS algorithm, there is no assumption on whether the server rate is constant or variable. Since an internal node in a hierarchical server is a variable rate server, this ensures that H-GPS is properly defined. Second, while $\phi's$ can be arbitrary positive numbers, it is the relative ratio's among them rather than the exact numbers that are important. For example, given an arbitrary set of $\phi's$, we can define a new set of $\hat{\phi}$'s that are normalized with respect to the sum of all $\phi's$, i.e., $\hat{\phi} = \frac{\phi_i}{\sum_{i=1}^{N}\phi_i}$. The GPS systems defined by ϕ's and $\hat{\phi}$'s are identical. Without losing generality, we assume that $\sum_{i=1}^{N}\phi_i = 1$ holds.

From (1), it immediately follows that

$$\frac{W_i(t_1,t_2)}{\phi_i} = \frac{W_j(t_1,t_2)}{\phi_j} \tag{2}$$

holds for any interval $[t_1, t_2]$ during which queues i and j are continuously backlogged. That is, the server services all backlogged sessions simultaneously, in proportion to their service shares. In addition

$$W_i(t_1,t_2) \geq \phi_i W(t_1,t_2) \tag{3}$$

holds for any interval $[t_1, t_2]$ during which queue i is continuously backlogged, i.e., queue i gets a minimum share of the server's capacity during any of its backlogged period *regardless of the behaviors of other sessions.* In the special case of a fixed-rate server with rate r, i.e., $W(t_1,t_2) = r(t_2 - t_1)$, (3) becomes

$$W_i(t_1,t_2) \geq r_i(t_2 - t_1) \tag{4}$$

where $r_i = \phi_i r$ is the minimum rate guaranteed to the session. With such a strong bandwidth guarantee, GPS can also provide a worst-case delay bound for a session that is constrained by a leaky bucket with an average rate no greater than r_i [12].

A good packet approximation algorithm of GPS would be one that serves packets in increasing order of their finish times in the fluid GPS system [5, 12]. However, when the packet system is ready to choose the next packet to transmit, it is possible the next packet to depart under the fluid

	Measured by Seconds	Measured by Bits
$V(t)$	$\int_{t_0}^{t} \frac{1}{\sum_{i \in B_{GPS}(\tau)} \phi_i} d\tau$	$\int_{t_0}^{t} \frac{r(t)}{\sum_{i \in B_{GPS}(\tau)} \phi_i} d\tau$
S_i^k	$max\{F_i^{k-1}, V(a_i^k)\}$	$max\{F_i^{k-1}, V(a_i^k)\}$
F_i^k	$S_i^k + \frac{L_i^k}{r_i}$	$S_i^k + \frac{L_i^k}{\phi_i}$

Table 1: Calculation of Virtual Times

system *have not arrived at the packet system.* Waiting for it requires knowledge of the future and also causes the system to be non-work-conserving. To have a work-conserving packet system, the packet server must choose a packet to transmit based only on the state of the fluid system up to time τ. In Weighted Fair Queueing (WFQ) [5], when the server is ready to transmit the next packet at time τ, it picks, among all the packets queued in the system at τ, the first packet that would complete service in the corresponding GPS system if no additional packets were to arrive after time τ. Since packet finish times will change when sessions become backlogged or unbacklogged, a naive implementation of WFQ is to re-compute the finish times of *all* packets in the GPS system whenever a session becomes backlogged or unbacklogged.

By observing the following important property of GPS [12], a more practical implementation of WFQ is possible.

Property 1 *The relative finish order of all packets that are in the system at time τ is independent of any packet arrivals to the system after time τ. That is, for any two packets p and p' at time τ in a GPS system, if p finishes service before p' assuming there are no arrivals after time τ, p will finish service before p' for any pattern of arrivals after time τ.*

With this property, it is possible to maintain the relative GPS finish order for packets in the WFQ system by using a priority queue mechanism [5, 12]. Such an implementation is based on the notion of a system virtual time function $V(\tau)$, which is the normalized fair amount of service that all backlogged sessions should receive by time τ in the GPS system. Each packet p_i^k (k^{th} packet on session i) has a virtual start and finish time S_i^k and F_i^k, where $V^{-1}(S_i^k)$ and $V^{-1}(F_i^k)$ are the times packet p_i^k starts and finishes services in the GPS system respectively. Another way of interpreting F_i^k is that it represents the amount of service, normalized with respect to its service share, session i has received right after packet p_i^k is served. In the GPS system, all backlogged sessions should receive the same normalized amount of service. Since both the system virtual time and the per packet virtual start/finish times represent the normalized amount of service, they are measured in unit of bits. In the special case of a fixed rate server, the elapsed time of a backlogged period is also a measure of the service provided by the server, therefore, virtual times can also be measured in unit of seconds. The exact algorithm for computing virtual times are shown in Table 1, where $B_{GPS}(\tau)$ is the set of backlogged queues at time τ, t_0 is the beginning of the system backlogged period that includes t, $r(\tau)$ is the server rate at time τ, and a_i^k and L_i^k are the arrival time and the length of packet p_i^k respectively. Notice that the definition of virtual times in unit of bits is more general, and is applicable to both fixed-rate and variable-rate servers.

Based on Property 1, for all packets present in the packet system at time τ, their relative finish order in GPS is the same as the relative order of their virtual finish times. Therefore, WFQ can be implemented by the "Smallest virtual Finish time First" (SFF) policy: when the server selects the next packet for service at time τ, it picks the packet with the smallest virtual finish time. An important advantage of this virtual-time-function-based implementation is that the virtual finish time of a packet can be computed at the packet arrival time and need not to be re-computed even if the set of backlogged sessions change in the future. The system virtual time function, though, does need to be re-computed when the set of backlogged sessions change. Since there can be N sessions that become backlogged or unbacklogged during an arbitrarily small interval, the worst case complexity of computing $V_{GPS}(.)$ is O(N) [7]. It is possible to have other Packet Fair Queueing (PFQ) algorithms based on virtual time functions with lower worst-case complexity [7, 8, 17]. Later in this paper, we will propose a more accurate virtual time function with a worst-case complexity of O(log N). Therefore, by exploiting Property 1 of GPS, it is possible to design virtual-time-function-based PFQ algorithms that have an overall complexity of O(log N).

2.2 H-GPS

A H-GPS server can be represented by a tree with a positive number ϕ_n associated with each node n. The root node, denoted by R, corresponds to the physical link and each leaf node corresponds to a session with a queue of packets. A non-leaf node is called backlogged if at least one of its leaf descendent nodes is backlogged. Let $W_i(t_1, t_2)$ be the amount of session i traffic served in the interval $[t_1, t_2]$, and $W_n(t_1, t_2) = \sum_{i \in leaf(n)} W_i(t_1, t_2)$, where $leaf(n)$ is the set of the leaf descendent nodes for node n. Also, for any node n, let $p(n)$ and $child(n)$ denotes its parent node and set of child nodes respectively. A work-conserving H-GPS server is defined as one for which

$$\frac{W_m(t_1, t_2)}{\phi_m} = \frac{W_{p(m)}(t_1, t_2)}{\sum_{q \in B_{p(m)}(t_1)} \phi_q} \tag{5}$$

holds for any interval $[t_1, t_2)$ during which node m is continuously backlogged and $B_{p(m)}(\tau)$, the set of backlogged child nodes of $p(m)$ at time τ, does not change. It immediately follows that

$$\frac{W_m(t_1, t_2)}{\phi_m} = \frac{W_n(t_1, t_2)}{\phi_n} \tag{6}$$

holds for any interval $[t_1, t_2]$ during which two sibling nodes m and n are continuously backlogged.

Assuming $\sum_{i \in leaf(R)} \phi_i = 1$ and $\sum_{m \in child(n)} \phi_m = \phi_n$, it can be shown that (3) holds also for H-GPS. Therefore, H-GPS can provide the same minimum bandwidth and delay bound guarantees for each session as GPS. The major difference between GPS and H-GPS is that while (2) holds for any two queues in GPS, (6) holds only for sibling nodes in H-GPS. Since in H-GPS packet queues are associated with only leaf nodes, the bandwidth is not always distributed to all queues in proportion to their service shares as in GPS. When a session cannot fully utilize its share of the service, the excess service is distributed according to the hierarchy.

H-GPS is also defined in a fluid system, therefore needs to be approximated by a packet algorithm. While it is possible to design practical packet approximation algorithms for GPS based on a single system virtual time function, this is not the case with H-GPS. The main reason is that Property 1 does not hold for H-GPS, i.e., *with H-GPS, the relative order of packet finish times is dependent on future arrivals.*

Consider the example where the root of H-GPS has two children A and B with service shares of 0.8 and 0.2 respectively. Node B is a leaf node while node A has two child leaf nodes A1 and A2 with service shares of 0.75 and 0.05. Let the link speed be 1 and all packets have the same length of 1. At time 0, A1 has an empty queue, A2 and B have many packets queued. Thus, A2 and B will have 80% and 20% of the link bandwidth respectively. With the assumption that there are no future arrivals, the finish times in the H-GPS server are 1.25, 2.5, 3.75, ..., for A2 packets, and 5, 10, 15, ..., for B packets. Therefore, at time 0, the relative order of packets is: p^1_{A2}, p^2_{A2}, p^3_{A2}, p^4_{A2}, p^1_B, p^5_{A2}, ... Now assume that a sequence of A1 packets arrive at time 1. According to the bandwidth distribution hierarchy, the bandwidth shares for A1, A2, B will be 75%, 5%, and 20% respectively. While this will not affect the finish times for session B packets, it does affect the finish times for session A2 packets. Between time [0,1], only 80% of the first packet of session A2 has been served. The rest of the 20% of the first packet and all the remaining packets will be served at 5% of the link rate. Therefore, the finish times are 5, 25, 45, 65, $\cdots$. That is, the relative ordering between session A2 and B packets have changed after the arrival of session A1's packets.

In a GPS system, during any period two sessions are both backlogged, the ratio between the services they receive is a constant regardless of future packet arrivals of other sessions. In a H-GPS system, this ratio is affected by other sessions in the hierarchy. This is the fundamental reason why Property 1 does not hold for H-GPS.

Without the relative-packet-order-invariant property, the concept of packet virtual finish times is not applicable. Therefore, the technique based on a single system virtual time function to approximate GPS does not apply to H-GPS.

3 H-PFQ

In this paper, we propose to approximate H-GPS by using one-level PFQ servers as basic building blocks and organizing them in a hierarchical structure. We call the resulted algorithms Hierarchical Packet Fair Queueing (H-PFQ).

A PFQ server node in a hierarchy differs from a standalone PFQ server in two aspects: it is a variable-rate server and the queues it serves do not have to be FIFO. As we discussed in the previous section, both GPS and WFQ are properly defined as variable-rate servers. In fact, by computing the system virtual time and per packet virtual start/finish times in unit of bits, most of the PFQ algorithms proposed in the literature [1, 7, 8, 13, 17] are variable-rate servers. Therefore, for PFQ nodes in an H-PFQ hierarchy, the virtual times should be measured in unit of bits.

The second difference between a standalone server and a server node in a hierarchy is that a standalone server serves per session *FIFO* queues whereas a server node serves per subtree logical queues that are *not* necessarily FIFO. A number of operations in the implementation of PFQ servers need to use packets from the head of each queue. While it is obvious which packet is at the head in a FIFO queue, we need to define the head packet for the logical queue that is associated with a child subtree.

In the following, we present an implementation framework of H-PFQ where an internal server node can be *any* PFQ algorithm that is properly defined as a variable rate server. The main data structure is a tree representation of the hierarchy. The root node represents the physical link and a leaf node represents a physical queue. Each non-root node n is connected to its parent $p(n)$ by a logical queue Q_n.

$V_n(t)$	the system virtual time function for node n
ϕ_n	service share for node n.
Q_n	the logical queue for node n
$Q_n(t)$	the packet at head of Q_n at time t
$\widehat{Q}_i$	the real queue for the leaf node i
$\widehat{Q}_i(t)$	the packet at head of $\widehat{Q}_i$ at time t
$s_n(t)$	the virtual start time of the packet $Q_n(t)$
$f_n(t)$	the virtual finish time of the packet $Q_n(t)$
$L_n(t)$	the length of the packet $Q_n(t)$
$Busy_n(t)$	true if node n is backlogged at t
$p(n)$	parent node of node n

Table 2: Notations used in the section

For the parent node to implement a PFQ algorithm, only the head of the logical queue is needed. Therefore, at any given time, only the reference to the packet, which is the head of the logical queue, is stored in queue Q_n. The actual packet remains stored in the real queue at the leaf node until the link finishes transmission of the packet. For consistency, we also define Q_R for the root server to be the packet currently being transmitted. At any given time when the server is busy, there exists a path from a leaf to the root such that the logical queues of all nodes traversed by the path point to the same physical packet that is currently being transmitted. The logical queues and associated data structures at each node are updated when a packet arrives at an empty session queue at the leaf node, or when the link is picking the next packet to transmit. In the following, we present the pseudocode to describe the details of the algorithm.

ARRIVE(i, $Packet$)
ENQUEUE($\widehat{Q}_i$, $Packet$)
if $Q_i(t) \neq \emptyset$
  then return
$Q_i(t) \leftarrow Packet$
$s_i(t) \leftarrow \max(f_i(t), V_{p(i)}(t))$
$f_i(t) \leftarrow s_i(t) + \frac{L_i(t)}{\phi_i}$
if $Busy_{p(i)} = FALSE$
  then RESTART-NODE($p(i)$)

When a packet arrives at a leaf node i, if session i's logical queue for its parent node Q_i is not empty, the packet is just appended to the end of the physical FIFO queue for the session. Otherwise, the packet also becomes the head of the logical queue Q_i. The virtual start and finish times for the logical queue are then updated, and the procedure *Restart-Node*() is called with the parent node if it is currently idle.

RESTART-NODE(n)
$m \leftarrow$ SELECT-NEXT(n)
if $m \neq \emptyset$
  then
    $ActiveChild_n \leftarrow m$
    $Q_n(t) \leftarrow Q_m(t)$
    if $Busy_n(t) = TRUE$
      then $s_n(t) \leftarrow f_n(t)$
      else $s_n(t) \leftarrow \max(f_n(t), V_{p(n)}(t))$
    $f_n(t) \leftarrow s_n(t) + \frac{L_n(t)}{\phi_n}$
    $Busy_n \leftarrow TRUE$
    UPDATE-V(n)
  else
    $ActiveChild_n \leftarrow \emptyset$
    $Busy_n \leftarrow FALSE$
if ($n \neq R$) *and* ($Q_{p(n)}(t) = \emptyset$)

then RESTART-NODE($p(n)$)
if $(n = R)$ *and* $(Q_R(t) \neq \emptyset)$
then TRANSMIT-PACKET-TO-LINK($Q_n(t)$)

A node is restarted whenever it needs to select a new packet to transmit. This occurs either when a packet arrives to an idle node or when the last packet finishes being transmitted on the physical link. If a packet arrives at an idle node n, the $Busy_n$ flag will be FALSE, in which case the new start time for the node is computed using $s_n(t) = \max(f_n(t), V_q(t))$. If the node is not idle and has a packet to transmit, the new start time will be set to the previous finish time. If the node has no more packets to send, the busy flag will be cleared. If the current node is not the root node, and its parent node does not have a packet in its logical queue Q_q, the node will restart its parent node. If the current node is the root node and there is a packet in the queue, the packet will be transmitted over the link. Different PFQ algorithms have different packet selection and system virtual time updating algorithms. For example, SFQ uses the Smallest virtual Start time First (SSF) policy and updates the system virtual time based on the virtual start time of the packet currently being served, whereas SCFQ uses the Smallest virtual Finish time First (SFF) policy and updates the system virtual time based on the virtual finish time of the packet currently being served. These algorithms are implemented in functions `Select-Next` and `Update-V`.

RESET-PATH(n)
$Q_n(t) \leftarrow \emptyset$
if $Leaf(n) = TRUE$
  then
    DEQUEUE($\widehat{Q}_n$)
    if $\widehat{Q}_n(t) \neq \emptyset$
      then
        $Q_n(t) \leftarrow \widehat{Q}_n(t)$
        $s_n(t) \leftarrow f_n(t)$
        $f_n(t) \leftarrow s_n(t) + \frac{L_n(t)}{r_n}$
    RESTART-NODE($p(n)$)
  else
    $m \leftarrow ActiveChild_n$
    $ActiveChild_n \leftarrow \emptyset$
    RESET-PATH(m)

When the link finishes transmitting a packet, it calls Reset-Path(R). Reset-Path descends the tree along the path to the leaf node whose packet just finished transmission. At each node along the path, it resets the logical queue to be empty. When the leaf node is reached, the first packet of the queue is dequeued and its parent node is restarted. During the descent, all pointers are cleared, but not the busy flags. During the process of picking a new packet, the busy flag acts as a reminder to the Restart-Node function that a packet has just finished transmission. If there are no more packets for this node to send, Restart-Node will clear the busy flag.

4 Delay Analysis of H-PFQ

In the previous section, we presented an algorithm to implement PFQ by integrating one-level PFQ's into a hierarchy. While most of the PFQ algorithms proposed in the literature can be used for this purpose, the delay bounds provided by the resulted H-PFQ servers can vary significantly with different PFQ algorithms.

In this section, we first give an example to show that with most of the PFQ algorithms proposed in the literature the resulted H-PFQ servers provide much larger delay bounds than those by H-GPS. We then present the concept of Worst-case Fair Index (WFI) and demonstrate that delay bounds provided by an H-PFQ server relates not only to delay bounds provided by PFQ server nodes in the hierarchy, but also to WFI's provided by the PFQ servers. In particular, in order to achieve tight delay bounds for H-PFQ, PFQ server nodes in the hierarchy need to have small WFI's. WF^{2}Q is the only algorithm proposed in the literature that provides tight WFI's, however it has a relatively high complexity. We propose a new algorithm called WF^{2}Q+ that not only provides tight delay bounds and low WFI's, but also has a relatively low complexity.

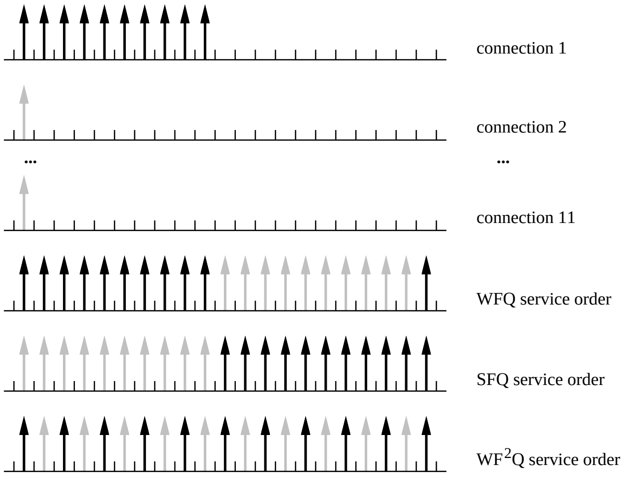


Figure 2: WFQ, SFQ, and WF^{2}Q

4.1 Limitation of Existing PFQ Algorithms

In [1], the following example is used to illustrate the large discrepancies between the services provided by GPS and WFQ. Assume that there are 11 sessions with packet size of 1 sharing a link with the speed of 1, $\phi_1 = 0.5$, and $\phi_i = 0.05, i = 2, \cdots, 11$. Session 1 sends 11 back-to-back packets starting at time 0 while each of all the other 10 sessions sends only one packet at time 0. If the server is GPS, it will take 2 time units to transmit each of the first 10 packets of session 1, one time unit to transmit the 11^{th} packet, and 20 time units to transmit the first packet from each of the other sessions. Denote the k^{th} packet of session j to be p_j^k, then in the GPS system, the finish time is $2k$ for $p_1^k, k = 1 \ldots 10$, 21 for p_1^{11}, and 20 for $p_j^1, j = 2, \cdots, 11$. Under WFQ, packets will be transmitted according to their finish times in the GPS system. Therefore, the first 10 packets of session 1 ($p_1^k, k = 1 \ldots 10$) will be transmitted, followed by one packet from each of sessions $2, \cdots, 11$ ($p_j^1, j = 2, \cdots, 11$), and then the 11^{th} packet of session 1 (p_1^{11}). In the example, between time 0 and 10, WFQ serves 10 packets from session 1 while GPS serves only 5. After such a period, WFQ needs to serve other sessions in order for them to catch up. Intuitively, the difference between the amounts of service provided to each session by WFQ and GPS is a measure of inaccuracy of WFQ in approximating GPS. In this case, the inaccuracy is $(N-1)/2$ packets, where N is the number of sessions sharing the link.

Such an inaccuracy introduced by WFQ will significantly affect the delay bound provided by H-WFQ. Consider the example with a link sharing structure in Fig. 1 (a) and the packet arrival sequence in Fig. 2. Assume that WFQ is used instead of GPS and the first 10 packets of class A1 belong to the best-effort sub-class and the 11^{th} packet belong to the real-time sub-class. Even though the real-time sub-class of A1 reserves 30% of the link bandwidth, when a real-time packet arrives, it may still have to wait 10 packet transmission times. Now consider the example where there are 1001 classes sharing a 100 Mbps link with the maximum packet size of 1500 bytes. For a real-time session reserving 30% of the link bandwidth, its packet may be delayed by 120 ms in just one hop! In contrast, if GPS or H-GPS is used, the worst-case delay for a packet arriving at an empty A1 real-time queue is 0.4 ms. Similar examples can be constructed for SCFQ [7], SFQ [8], and FBFQ [17].

4.2 WFI and Its Effect on Delay Bounds of H-PFQ

In [1], we introduce a metric called Worst-case Fair Index (WFI) to characterize PFQ servers. In this section, we will develop analysis techniques to show that delay bounds provided by an H-PFQ server relates not only to delay bounds provided by PFQ server nodes in the hierarchy, but also to WFI's provided by the PFQ servers. In particular, in order to achieve tight delay bound for H-PFQ, PFQ server nodes in the hierarchy need to have small WFI's.

Definition 1 *A server s is said to guarantee a Time Worst-case Fair Index (T-WFI) of $\mathcal{A}_{i,s}$ for session i, if for any time τ, the delay of a packet arriving at τ is bounded above by $\frac{1}{r_i}Q_i(\tau) + \mathcal{A}_{i,s}$, that is,*

$$d_i^k - a_i^k \leq \frac{Q_i(a_i^k)}{r_i} + \mathcal{A}_{i,s} \tag{7}$$

where r_i is the rate guaranteed to session i, $Q_i(\tau)$ is the number of bits in the session queue at time τ (including the packet that arrives at time τ), a_i^k and d_i^k are the arrival and departure times of the k^{th} packet of session i respectively.

For the purpose of this paper, a packet is said to arrive or leave the server if its last bit arrives or leaves the server. Intuitively, $\mathcal{A}_{i,s}$ represents the maximum time a packet coming to an empty queue needs to wait before receiving its guaranteed service rate. An important observation is that both GPS and H-GPS have a WFI of 0. That is, with GPS or H-GPS, a packet coming to an empty queue can receive its guaranteed service rate immediately after its arrival. However, as illustrated in the example in Fig. 2, the T-WFI for WFQ can increases linearly as a function of the number of sessions N.

Since the previous definition of WFI applies only to a standalone server, which is fixed-rate and has only one level, we introduce a general definition of WFI that applies also to server nodes in a hierarchy, which are variable rate servers. Again as in Sections 2 and 3, we generalize the definition by measuring WFI in unit of bits instead of seconds.

Definition 2 *A server node s is said to guarantee a Bit Worst-case Fair Index (B-WFI) of $\alpha_{i,s}$ for session i, if for any packet p_i^k the following holds*

$$W_i(t_1, d_i^k) \geq \frac{\phi_i}{\phi_s} W_s(t_1, d_i^k) - \alpha_{i,s} \tag{8}$$

where d_i^k is the time p_i^k departs the server, t_1 is any time instant such that $t_1 < d_i^k$ and session i is continuously backlogged during $[t_1, d_i^k]$, and $\frac{\phi_i}{\phi_s}$ is the service share guaranteed to queue i by server s.

For a constant rate one-level server, $\phi_s = 1$, and $W_s(t_1, t_2) = r_s(t_2 - t_1)$. Therefore, (8) is equivalent to:

$$W_i(t_1, t_2) \geq r_i(t_2 - t_1) - \alpha_{i,s} \tag{9}$$

In this case, Definition 2 subsumes Definition 1 and $\alpha_{i,s} = r_i \mathcal{A}_{i,s}$ holds. This can be easily established by letting $t_1 = a_i^k$ and using the following property for a FIFO queue i

$$W_i(a_i^k, d_i^k) = Q_i(a_i^k) \tag{10}$$

Before we proceed to establish the relationship between the delay bound of an H-PFQ server and WFI's of PFQ server nodes, we first give the following definition of guaranteed service burstiness index (SBI), which is a generalized bounded delay property that applies to both constant-rate and variable-rate servers.

Definition 3 *A server s is said to guarantee a service burstiness index (SBI) of $\gamma_{i,s}$ to session i if for any packet p_i^k, there exists a time instant t_1 within the server's busy period that includes also d_i^k, where $t_1 < d_i^k$, $Q(t_1^-) = 0$, and $Q(t_1) \neq 0$, such that*

$$W_i(t_1, d_i^k) \geq \frac{\phi_i}{\phi_s} W_s(t_1, d_i^k) - \gamma_{i,s} \tag{11}$$

holds where d_i^k is the time p_i^k departs the server and $\frac{\phi_i}{\phi_s}$ is the service share guaranteed to queue i by server s.

The definition of SBI has its root in the guaranteed service curve concept proposed by Cruz [3]. However, there are several differences. First, SBI is applicable to both constant-rate and variable-rate servers, while guaranteed service curve is defined only for constant-rate servers. Second, for SBI, we consider only intervals that end at packet departure times, while Cruz considers intervals that end at arbitrary time instants. Since both SBI and guaranteed service curve are used to reason a session's delay property, considering only time intervals that end at packet departure times will result in a tighter bound. Finally, in the definition of SBI, we require that t_1 and d_i^k be within the same *system* busy period (but not necessarily in the same session i's backlogged period), while Cruz's definition does not have such a requirement. Therefore, SBI represents a stronger guarantee than the guaranteed service curve. However, it can be shown that all the analysis and results in [3] are applicable with our definition of SBI. Intuitively, for any work-conserving queueing system (including PFQ systems), system busy periods are invariant with respect to the scheduling policy used, therefore can be independently analyzed.

While the definitions of WFI and SBI look similar, worst case fairness is a stronger property than bounded service burstiness. In the case of the worst-case fair property, (11) needs to hold for *all intervals* that ends with d_i^k and during which session i is continuously backlogged. In the case of the guaranteed service burstiness property, (8) needs to hold for only *one* interval that ends at d_i^k and starts at the beginning of a session i's backlogged period. By letting t_1 to be the start of the backlogged period that includes d_i^k, it immediately follows that *a session's guaranteed WFI is also the session's guaranteed SBI.* The opposite is not always true.

For example with WFQ, the guaranteed SBI for any session is P_{max}. This is much smaller than the guaranteed WFI value, which can be as large as $N \times P_{max}$.

In the following lemma, we establish the relationship between the guaranteed SBI and guaranteed delay bound to a leaky bucket constrained session. A session i is constrained by a leaky bucket (σ_i, ρ_i) if the following holds for any interval $[t_1, t_2]$

$$A_i(t_1, t_2) \leq \sigma_i + \rho_i(t_2 - t_1) \tag{12}$$

where $A_i(t_1, t_2)$ is the amount of session i bits arrived during $[t_1, t_2]$.

Lemma 1 *Consider session i that is leaky bucket constrained by (σ_i, r_i). If a standalone server with a constant rate r guarantees an SBI of $\gamma_{i,s}$ to session i, it can guarantee a delay bound of $\frac{\sigma_i+\gamma_{i,s}}{r_i}$ where $r_i = \frac{\phi_i}{\phi_s} r$.*

Proof. For a constant rate server, $W_s(t_1, t_2) = r(t_2 - t_1)$ hold during any system backlogged period. Since the server guarantees an SBI to session i, there exists a time instant t_1, where $t_1 < d_i^k$, $Q_i(t_1^-) = 0$, and $Q_i(t_1) \neq 0$ hold, such that

$$\begin{aligned} W_i(t_1, d_i^k) &\geq \frac{\phi_i}{\phi_s} W_s(t_1, d_i^k) - \gamma_{i,s} \\ &= r_i(d_i^k - t_1) - \gamma_{i,s} \end{aligned} \tag{13}$$

Since session i has a FIFO queue and $Q_i(t_1^-) = 0$, we have

$$W_i(t_1, d_i^k) = A_i(t_1, a_i^k) \tag{14}$$

In addition, session i is leaky bucket constrained, thus

$$A_i(t_1, a_i^k) \leq \sigma_i + r_i(a_i^k - t_1) \tag{15}$$

Combining (13), (14), and (15), we have

$$\sigma_i + r_i(a_i^k - t_1) \geq r_i(d_i^k - t_1) - \gamma_{i,s} \tag{16}$$

Rearranging terms and dividing both sides by r_i, we have

$$d_i^k - a_i^k \leq \frac{\sigma_i + \gamma_{i,s}}{r_i} \tag{17}$$

Q.E.D.

For most rate-based service disciplines [20], if the server guarantees a delay bound of D_i to a leaky bucket constrained session, it also guarantees an SBI of $r_i D_i - \sigma_i$ to the session [3]. Therefore, bounded service burstiness property and bounded delay property are equivalent for standalone rate-based servers. Since the bounded service burstiness property applies also to variable rate servers, it can be viewed as the generalized bounded delay property.

From Lemma 1, it immediately follows that a bounded WFI for a session also implies a bounded delay. However, the delay bound calculated from the WFI may not be tight in some cases. For example, while the tight delay bound for a leaky bucket constrained session in a WFQ server is $\frac{\sigma_i}{r_i} + \frac{P_{max}}{r}$, the delay bound based on the WFI can be $\frac{\sigma_i}{r_i} + N\phi_i \frac{P_{max}}{r}$, which is much larger. Intuitively, WFI is the maximum amount of time a packet has to wait to receive its fair share service when it comes to an empty queue i. The reason that a packet may have to wait for a long time is that some packets *related to it* have received *more* service than deserved in a previous time period. In the case of a standalone server, these packets must belong to the same session i. In the case of a hierarchical server, these packets may belong to sessions that share an ancestor node with session i. Therefore, WFI does not bound delay tightly in the case of a standalone server since it does not take into account the fact that packets from the same session may receive more service in a previous time period. However, WFI is important in characterizing the delay in a hierarchical server since the extra service received in the previous time period may have been received by a session other than the one being considered.

Now that we have defined WFI and SBI that are applicable to both standalone servers and server nodes in a hierarchy, we are ready to derive WFI's and delay bounds provided by an H-PFQ server. For a session i with H ancestors in an H-PFQ server, we use $p(i)$ to represent its parent node, $p^h(i)$ to represent the parent node of $p^{h-1}(i)$ for $h = 1, \cdots, H$, where $p^0(i) = i$, $p^1(i) = p(i)$, and $p^H(i) = R$.

Theorem 1 *For a session i with H ancestors in an H-PFQ server, it is guaranteed the following B-WFI*

$$\alpha_{i,H-PFQ} = \sum_{h=0}^{H-1} \frac{\phi_i}{\phi_{p^h(i)}} \alpha_{p^h(i)} \tag{18}$$

where $\alpha_{p^h(i)}$ is the B-WFI for the logical queue at node $p^h(i)$ for the server node $p^{h+1}(i)$, h=0, $\cdots$, H-1.

The proof is given in Appendix A. Basically, the theorem states that the WFI provided to a session by an H-PFQ server is the weighted sum of WFI's of all the session's ancestor servers.

Since a bounded WFI also implies a bounded delay, it immediately follows that

Corollary 1 *For a session i with H ancestors in an H-PFQ server, if it is constrained by a leaky bucket (σ_i, r_i), the delay of any packet in the session is bounded by*

$$\frac{\sigma_i}{r_i} + \sum_{h=0}^{H-1} \frac{\alpha_{p^h(i)}}{r_{p^h(i)}} \tag{19}$$

While Corollary 1 gives the delay bound for a leaky bucket constrained session in an H-PFQ server, the bound is not the tightest as it does not account for the situation where packets from the same session received more service in a previous time period. The following theorem provides a tighter bound.

Theorem 2 *For a session i with H ancestors in an H-PFQ server, if it is constrained by a leaky bucket (σ_i, ρ_i), the delay of any packet in the session is bounded by*

$$D_i + \sum_{h=1}^{H-1} \frac{\alpha_{p^h(i)}}{r_{p^h(i)}} \tag{20}$$

where $\alpha_{p^h(i)}$ is the B-WFI for the logical queue at node $p^h(i)$ for the server node $p^{h+1}(i)$, h=1, $\cdots$, H-1, and $r_i D_i - \sigma_i$ is the SBI guaranteed to session i by its parent server node, i.e., D_i is the delay bound guaranteed to session i by a standalone $p(i)$ server.

The proof of the theorem is given in Appendix B. The theorem states that the delay bound provided by an H-PFQ server to session i is the sum of the delay bound provided by session i's parent server to session i and the WFI's of all the other session i's ancestor nodes weighted by their guaranteed shares. This bound is tight when $\alpha_{p^h(i)}$'s and D_i are tight. This can be easily shown by constructing examples as in Section 4.1. Therefore, to achieve tight delay bounds in a H-PFQ server, the WFI's for the internal and root server nodes should be small.

5 Worst-case Fair PFQ Algorithms

5.1 WF^{2}Q

Among all PFQ algorithms proposed in the literature, the Worst-case Fair Weighted Fair Queueing (WF^{2}Q) [1] is the only one that provides tight WFI's.

WF^{2}Q differs from WFQ in that it uses the "Smallest Eligible virtual Finish time First" (SEFF) policy instead of the popular SFF or SSF policies. With WF^{2}Q, when the server picks the next packet to transmit at time τ, rather than selecting it from among all the packets at the server as in WFQ, the server only considers the set of packets that have started service in the corresponding GPS system, and selects the packet among them that has the smallest virtual finish time. A packet is said to be *eligible at time* τ if its virtual start time is no greater than the current system virtual time.

If we consider again the example in Section 4.1, at time 0, all packets at the head of each session's queue, p_i^1, $i = 1, \cdots, 11$, have started service in the GPS system. Among them, p_1^1 has the smallest finish time in GPS, so it will be transmitted first in WF^{2}Q. At time 1, there are still 11 packets at the head of the queues: p_1^2 and p_i^1, $i = 2, \cdots, 11$. Although p_1^2 has the smallest virtual finish time, it will not start service in GPS until time 2, therefore, it won't be eligible for transmission at time 1. The other 10 packets have all started service at time 0 at the GPS system, thus are eligible, and one of them will be transmitted. At time 3, p_1^2 becomes eligible and has the smallest finish time among all packets, thus it will be transmitted next. The service order for all packets under WF^{2}Q is shown as the last time line in Fig. 2. As can be seen in the example, during any time interval, the difference between the amounts of bits transmitted by GPS and WF^{2}Q is less than one packet size. Therefore, WF^{2}Q is a more accurate approximation of GPS than WFQ. The following theorem is proven in [1].

Theorem 3 *(1) WF^{2}Q is a work-conserving policy.*
(2) WF^{2}Q is worst-case fair for session i with the following worst-case fair index

$$\alpha_{i,WF^2Q} = L_{i,max} + (L_{max} - L_{i,max})\frac{r_i}{r} \qquad (21)$$

(3) For a session i constrained by a leaky bucket (σ_i, r_i), WF^{2}Q guarantees a delay bound of $\frac{\sigma_i}{r_i} + \frac{L_{max}}{r}$.

As can be seen, the WFI provided by WF^{2}Q is independent of the number of sessions sharing the server. In the case of $L_{i,max} = L_{max}$, α_{i,WF^2Q} will simply be L_{max}. Since the B-WFI for a packet system is at least one packet size, WF^{2}Q is an optimal packet policy with respect to the worst-case fair property. In addition, since the minimum difference between a delay bound provided by a PFQ server and a GPS server is one packet transmission time, both WFQ and WF^{2}Q provide the tightest delay bound among all PFQ algorithms.

5.2 WF^{2}Q+

While WF^{2}Q provides the tightest delay bound and smallest WFI among all PFQ algorithms, it has the same worst-case complexity of O(N) as WFQ because they both need to compute $V_{GPS}(\cdot)$.

In this section, we present a new packet algorithm that provides the same delay bound and WFI as WF^{2}Q, but with a lower complexity. Since this policy is also worst-case fair, but is simpler than WF^{2}Q, we call it WF^{2}Q+. WF^{2}Q+ also uses the SEFF policy. The novel aspect of WF^{2}Q+ is the use of a new system virtual time function $V_{WF^2Q+}(\cdot)$ that achieves both low complexity and high accuracy in approximating the ideal virtual time function used in GPS. While a number of new virtual time functions have been proposed to simplify the implementation of WFQ [7, 17], they all result in PFQ algorithms with large WFI's. The unique advantage of $V_{WF^2Q+}(\cdot)$ is that the resulted WF^{2}Q+ algorithm combines all three properties that are important for a PFQ algorithm to be used in a H-PFQ server: tight delay bound, small WFI, and low algorithmic complexity.

With WF^{2}Q+, the virtual time function is defined as

$$V_{WF^2Q+}(t+\tau) = \max(V_{WF^2Q+}(t) + W(t, t+\tau), \min_{i \in \hat{B}(t)} (S_i^{h_i(t)})) \qquad (22)$$

where $W(t, t+\tau)$ is the total amount of service provided by the server during the period $[t, t+\tau]$, $\hat{B}(t)$ is the set of sessions backlogged in the WF^{2}Q+ system at time t, $h_i(t)$ is the sequence number of the packet at the head of the session i's queue, and $S_i^{h_i(t)}$ is the virtual start time of the packet.

There are several noteworthy properties of $V_{WF^2Q+}(\cdot)$. First, if we view the system virtual time function as a function of the amount of service provided by the server, $V_{WF^2Q+}(\cdot)$ is a strictly monotonically increasing function of time with a minimum slope of 1. We call this the "minimum slope property" of V_{WF^2Q+}, which is important for a PFQ server to provide delay bounds to leaky bucket constrained sources that are within one packet transmission time of those provided by GPS. The virtual time function $V_{GPS}(\cdot)$, used by both WFQ and WF^{2}Q, has this property by using the marginal service rate of the GPS server as the slope, which has a minimum value of 1. Therefore, both WFQ and WF^{2}Q can provide tight delay bounds. On the other hand, the virtual time functions used by SCFQ [7] and SFQ [8] may have a slope of 0 during certain periods, and the delay bounds provided by the resulted SCFQ and SFQ algorithms are much larger than those provided by WFQ and WF^{2}Q. The second important property of $V_{WF^2Q+}(\cdot)$, as provided by the max over min operation in (22), is that it is at least as large as the minimum virtual start time of all packets at the head of all queues. This has two implications. First, this ensures that a newly backlogged session has a virtual start time at least as large as one of the existing backlogged sessions. This is important for the resulted WF^{2}Q+ algorithm to achieve a low WFI. In addition, the property also ensures that at least one packet in the system has a virtual start time no greater than the current system virtual time. This guarantees the resulted SEFF policy to be work-conserving as only packets with virtual start time no greater than the current system virtual time are eligible for transmission.

To simplify the implementation, we also modify the definition of virtual start and finish times. With the old definition as in Table 1, virtual start and finish times need to be maintained on a per packet basis. Usually this means stamping the values of S_i^k and F_i^k in the header of packet p_i^k. This overhead may not be acceptable for networks with small packet sizes, such as ATM networks. With the following definition, there is only one pair of F_i and S_i that needs to be maintained for each session i. Whenever a packet p_i^k reaches the head of the queue, F_i and S_i are updated according to the following

$$S_i = \begin{cases} F_i & \text{if } Q_i(a_i^k-) \neq 0 \\ max(F_i, V(a_i^k)) & \text{if } Q_i(a_i^k-) = 0 \end{cases} \tag{23}$$

$$F_i = S_i + \frac{L_i^k}{r_i} \tag{24}$$

where $Q_i(a_i^k-)$ is the queue size of session i just before time a_i^k. With this definition, per session S_i and F_i are also the virtual start and finish times of the packet at the head of the session queue.

There are two major tasks associated with implementing WF^{2}Q+: (a) computing the system virtual time function, and (b) maintaining the set of eligible sessions sorted by virtual finish times. Both can be accomplished with O(log N) complexity [19].

The delay and worse-case fairness properties of WF^{2}Q+ are given by the following theorem.

Theorem 4 *(1) WF^{2}Q+ is work-conserving.*
(2) WF^{2}Q+ is worst-case fair for session i with

$$\alpha_{i,WF^2Q+} = L_{i,max} + (L_{max} - L_{i,max})\frac{r_i}{r} \tag{25}$$

(3) For a session i constrained by a leaky bucket (σ_i, r_i), WF^{2}Q+ guarantees a delay bound of $\frac{\sigma_i}{r_i} + \frac{L_{max}}{r}$.

Since the proof is rather long, we will just present its outline in this paper. The full proof will appear in a follow-up paper. The proof is based on the theory of rate-proportional servers developed in [18].

While the definition of rate-proportional servers in [18] is based on virtual time functions measured in unit of second, it can be easily extended to the more general definition with virtual time functions measured in unit of bit. A fluid rate proportional server with N sessions is characterized by N numbers $\phi_1, \cdots, \phi_N$, and a system virtual time function, which must satisfy the following two conditions:

$$V(t_2) - V(t_1) \geq W(t_1, t_2) \tag{26}$$

$$V(t) \leq min_{i \in B(t)} V_i(t) \;\; \forall\, t \tag{27}$$

where $(t_1, t_2]$ is any interval in a system backlog period, $B(t)$ is the set of backlogged sessions in the fluid system at time t, and $V_i(\cdot)$ is the virtual time function for session i, which is iteratively defined as follows

$$V_i(t_2) = \begin{cases} V_i(t_1) \\ \quad if \;\; Q_i(\tau) = 0, \; \forall\, t_1 < \tau < t_2 \\ \\ max\{V(t_2), V_i(t_2-)\} \\ \quad if \;\; Q_i(t_2-) = 0 \; \wedge \; Q_i(t_2) \neq 0 \\ \\ V_i(t_1) + \frac{W_i(t_1,t_2)}{\phi_i} \sum_{j=1}^{N} \phi_j \\ \quad if \;\; Q_i(\tau) \neq 0, \; \forall\, t_1 \leq \tau \leq t_2 \end{cases} \tag{28}$$

At any given time, the server simultaneously service all sessions that have the minimum virtual time function, proportionally to their relative service shaes. Formally, during any time period $(t_1, t_2]$ in which, $C(\cdot)$, the set of backlogged sessions with the minimum virtual time function $V_i(\tau)$, is unchanged, the server services all sessions in $C(t_1)$ such that

$$\frac{W_i(t_1, t_2)}{\phi_i} = \frac{W(t_1, t_2)}{\sum_{j \in C(t_1)} \phi_j} \;\;\; \forall i \in C(t_1) \tag{29}$$

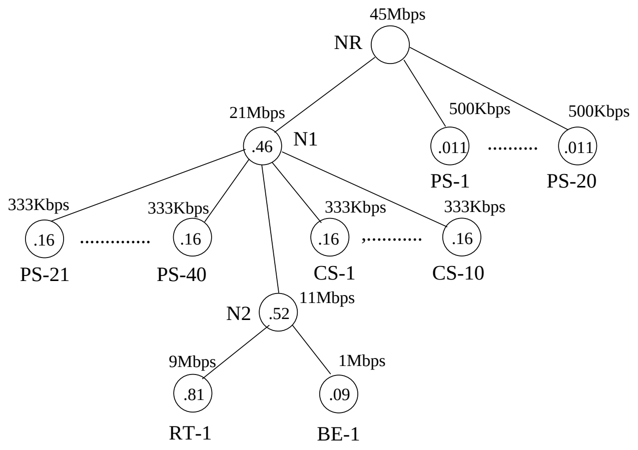


Figure 3: Example 1

It can be shown that GPS is a special rate-proportional server with the system virtual time function $V_{GPS}(\cdot)$. In fact, with GPS, $V_i(\tau) = V_j(\tau)$ holds for any two sessions backlogged at time τ, and therefore $C(\tau) = B(\tau)$ holds for any time instance τ.

For each fluid rate-proportional server, two PFQ algorithms can be defined based on the SFF and the SEFF packet selection policies. For example, for GPS, WFQ and WF^{2}Q are the corresponding PFQ algorithms with SFF and SEFF packet selection policies respectively. By applying similar techniques that are used in [1] to prove the properties of WF^{2}Q, the following theorem can be established.

Theorem 5 *For any rate proportional server, its corresponding PFQ algorithm with the SEFF policy can provide to session i*

1. *a delay bound of $\frac{\sigma_i}{r_i} + \frac{L_{max}}{r}$ if the session is constrained by a leaky bucket (σ_i, r_i), and*
2. *a WFI of $L_{i,max} + (L_{max} - L_{i,max})\frac{r_i}{r}$.*

In addition, the following theorem holds.

Theorem 6 *WF^{2}Q+ is a packet rate-proportional server with the SEFF packet selection policy.*

The proof is rather long and will be presented in a follow-up paper. The main results of Theorem 4 follow directly from Theorems 5 and 6. In addition, WF^{2}Q+ is work-conserving because there is at least one packet eligible for service during any system backlogged period.

Therefore, any PFQ algorithm that approximates a fluid rate-proportional server with SEFF policy achieves the same worst-case fairness and bounded delay properties as WF^{2}Q. The unique advantage of WF^{2}Q+ is that it uses a novel virtual time function with a lower complexity.

The Corollary below, which gives the delay bound for H-WF^{2}Q+, follows directly from Theorem 2 and Theorem 4.

Corollary 2 *For a session i with H ancestors in a H-WF^{2}Q+ server, if it is constrained by a leaky bucket (σ_i, ρ_i) and $L_{max} = L_{i,max}$, the delay of any packet in the session is bounded by*

$$\frac{\sigma_i}{r_i} + \sum_{h=0}^{H-1} \frac{L_{max}}{r_{p^h(i)}} \tag{30}$$

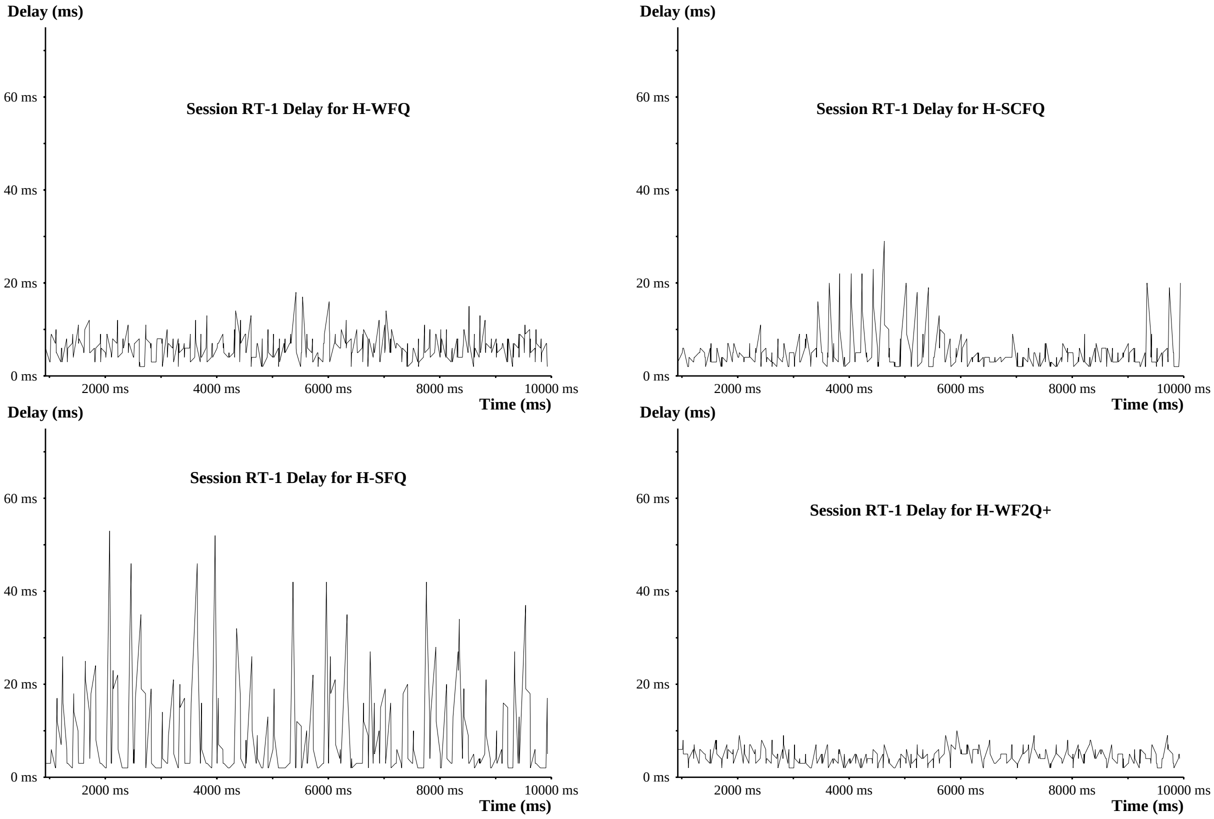


Figure 4: Delay of RT-1 With Uncorrelated Cross Traffic

6 Simulation Experiments

In this section, we present simulation experiments to illustrate the bounded delay and hierarchical link-sharing properties of H-WF^{2}Q+ . For the purpose of verifying, we had two independent implementations of all the algorithms in two different simulators. All experiments were conducted in both simulators and the results matched each other.

6.1 Delay Characteristics

In this section we compare the packet delay distributions for a real-time session under four different H-PFQ servers, H-WF^{2}Q+ , H-WFQ, H-SFQ, and H-SCFQ. The service hierarchy is shown in Fig 3. The rate above the node is the guaranteed service rate for the node. The value inside the node represents the node's service share with respect to its parent node.

The real-time session being measured is the leaf node labeled RT-1 . It has a guaranteed service share of 0.81 from its parent node which translates into a guaranteed rate of 9 Mbps. Session RT-1 is a deterministic on/off source with a 25 ms on-period and a 75 ms off-period. Session RT-1 has a continuously backlogged sibling session BE-1 . As a result, nodes N1 , N2 , and NR are also continuously backlogged. We use two additional types of background traffic: poisson sources that are labeled PS-n and constant rate sessions that are labeled CS-n . All constant rate sessions have identical start times and a peak transmission rate equal to their guaranteed rate. They first passed through a multiplexer before they arrive at the server, so that they do not have simultaneous arrivals, but rather model the sort of packet train burst that could be sent by individual users and/or networks with high speed connections. For simplicity, we assume all sessions transmit 8 KB packets.

We consider two scenario's: (a) uncorrelated cross traffic, and (b) correlated cross traffic.

Fig. 4 shows the case when PS-n sources are transmitting at an average of 1.5 times their guaranteed rate and the constant rate sources are not transmitting. As a result all the PS-n sessions eventually become persistently backlogged. As we can see, while on average the delays for all four algorithms are similar, the worst case packet delays under H-SFQ, H-SCFQ, and H-WFQ are larger than those under H-WF^{2}Q+ .

For experiments shown in Fig. 5, everything remains the same except that the correlated constant rate sessions are turned on. As can be seen, the worst-case delay increases substantially under H-WFQ, H-SFQ, and H-WFQ, but remains almost the same for H-WF^{2}Q+ . H-SFQ is affected most by the presence of correlated traffic. This can be understood by the following intuitive explanation. Since the packets are well spaced out for constant rate sessions, they usually arrive at an empty session queue. With SFQ, these packets will be assigned the same virtual start time as the packet currently being served, which has the *smallest* virtual start times among all backlogged packets. Since SFQ picks the packet with the smallest virtual arrival times, the newly arrived packets will be served ahead of other backlogged packets, including RT-1 packets. If there is a burst of packet arrivals into empty queues for a short interval, the system virtual time will stop advancing, and only start advancing again after these packets finish service. As a result, the delay for other packets in the system will increase. This can be seen also from the example illustrated in Fig. 2.

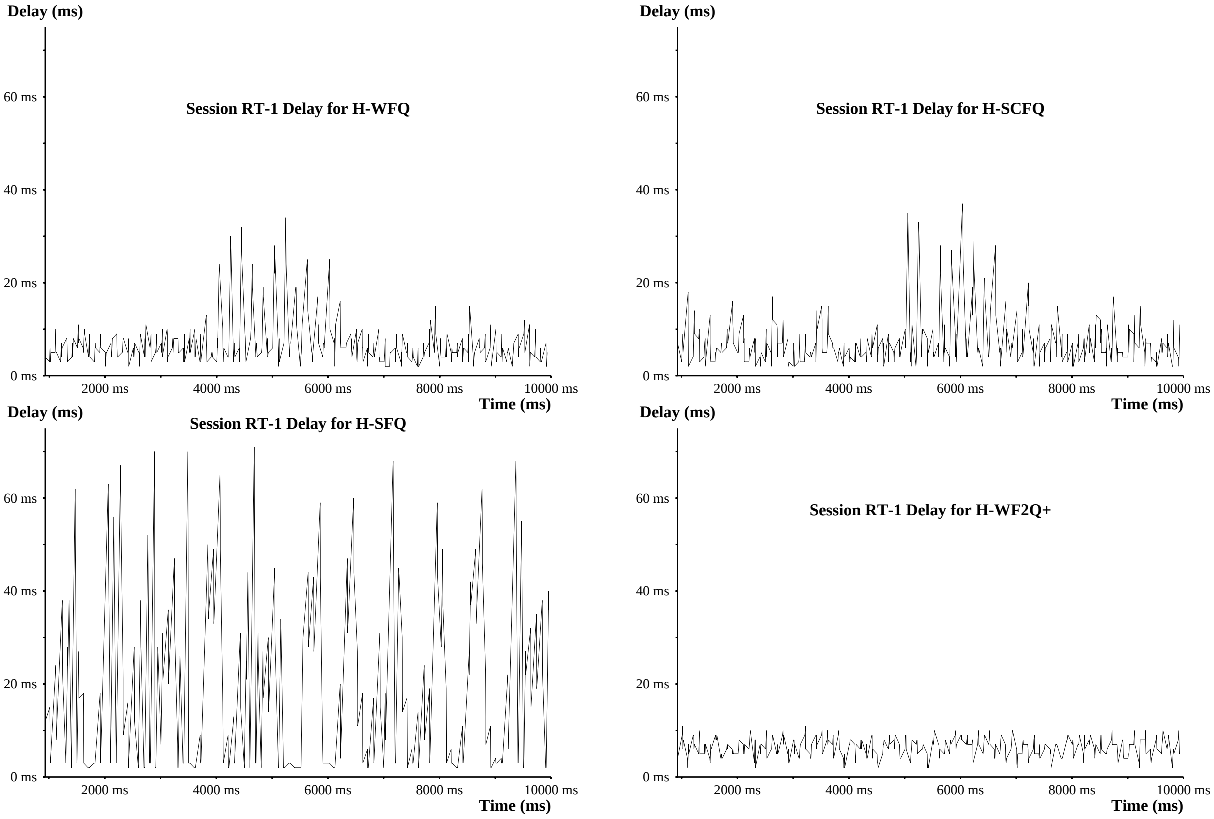


Figure 5: Delay of RT-1 With Correlated Cross Traffic

6.2 Hierarchical Link Sharing

We consider the link-sharing structure shown in Fig. 6(a), which has a multi-level hierarchy with two types of sources: TCP sources and deterministic on-off sources. We will examine the performance of sessions labeled TCP-{1,5,8,10,11} under link-sharing when on-off sources alternate between active and idle states. To see the effect of hierarchical link-sharing, we use one on-off source for each level in the hierarchy. The bandwidth's and active periods of the on-off sources are shown in Fig. 6(b).

Fig. 7 shows the bandwidth vs. time plots for each of the TCP sessions under consideration. The bandwidth is measured by averaging over 100 ms windows, with two adjacent windows overlapping 50 ms. As can be seen, all four algorithms perform well. While 100 ms provides a very fine granularity of measuring bandwidth, it is a very *large* number when it comes to one hop average packet delay. Therefore, even though the worst-case packet delays vary significantly with different H-PFQ algorithms, the bandwidth distribution are very similiar.

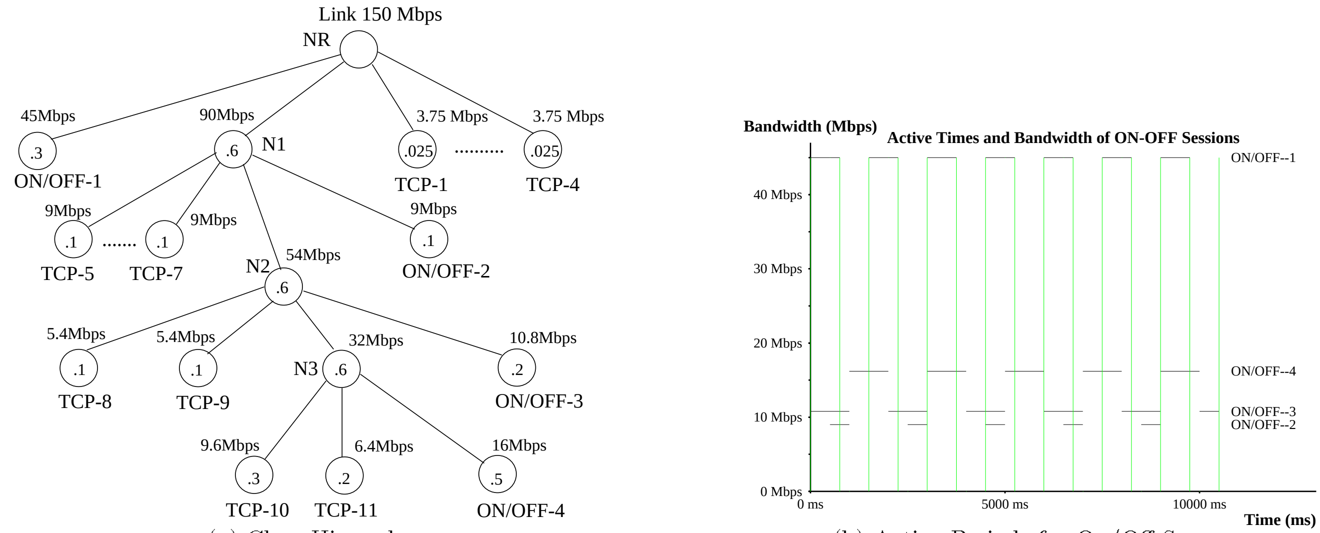
7 Related Work

In [15], H-WFQ is used to support integrated traffic management. The negative effects introduced by WFQ's high WFI on link-sharing and traffic management algorithms are not studied. To provide tighter bounds for real-time traffic, all real-time queues need to be children of the root node, and link-sharing between real-time and non-real-time sessions is accomplished via a separate mechanism.

In [11], an implementation of H-WFQ is presented. The scheduler implemented is actually not an H-WFQ server, but a WFQ server in which the weights are dynamically changed according to the set of backlogged sessions in the packet server. It is easy to show that such an implementation will not only yield much larger delay bounds but also violate the link-sharing goals in certain situations. The key problem is that at any time instance, the set of the backlogged sessions in a packet system can be quite different from that in the corresponding fluid system. Adjusting the weight according to the set of backlogged sessions in the packet system can result in large errors.

In [6], a Class-Based Queueing (CBQ) algorithm is presented to support link-sharing and integrated 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. Our work differs from this work in that we build our framework on H-GPS, which has theoretically proven properties for supporting link-sharing, real-time service, and best-effort service.

A number of algorithms such as Self-Clocked Fair Queueing [4, 7], Stochastic Fair Queueing [10], Deficit Round Robin [16], Frame-based Fair Queueing [17], and Start-time Fair Queueing [8] have been proposed to approximate GPS with a lower complexity. However, none of them address the issue of worst-case fairness, and all of them have large WFI's. As shown in the paper, H-PFQ algorithms based on these algorithms result in much larger worst-case delay than that under H-WF^{2}Q+. The Leap Forward Virtual Clock [13] achieves low WFI by using a SEFF policy similar to that



(a) Class Hierarchy (b) Active Periods for On/Off Sources

Figure 6: Class Hierarchy, ON/OFF Sources and Measured Bandwidth

used by WF^2Q and WF^2Q+. We hypothesize that it belongs to the class of PFQ algorithms that approximate a fluid rate proportional server using a SEFF policy.

The idea of implementing H-PFQ algorithm by integrating one-level PFQ algorithm into a hierarchy was also independently proposed in [8]. However, the details of the algorithm are not presented and the analysis applies only to H-SFQ. In addition, there are two claims made in [8] that we don't believe are accurate. In [8], it was claimed that SFQ had two unique advantages compared to other PFQ algorithms: first, it is the only algorithm that are fair when the server is variable rate, second, it is the only algorithm that does not require admission control (the sum of service shares does not need to be less than 1). As discussed in Section 2, by measuring virtual time functions in unit of bits instead of seconds, all existing PFQ algorithms are fair even when the server is variable rate, and therefore, they can all be used to implement H-PFQ algorithms. As shown in Section 6, even though the delay bounds provided by the resulted H-PFQ algorithms can vary significantly, the fairness (link-sharing) property is maintained by all algorithms. For admission control, as discussed in Section 2, it is the relative ratio's among session's service shares that are important. The exact values of the service shares are not important. Normalizing all service shares with respect to the sum of service shares will result in an identical policy as before. Admission control is required only for sessions that require *minimum* service guarantees – the total amount of services that are allocated to the sessions requiring performance guarantees should be less than the total server capacity. This condition needs to be held for *all* PFQ algorithms, including SFQ.

8 Conclusion

We have made several contributions in this paper. First, we proposed a formal model based on the idealized H-GPS system to simultaneously support guaranteed real-time, adaptive best effort, and controlled link-sharing services. Second, we presented an algorithm to implement H-PFQ by organizing one-level PFQ servers in a hierarchical structure. Most of PFQ algorithms can be used for this purpose. The key is to compute the system virtual time and per packet virtual start/finish times in unit of bits instead of seconds. Third, we develop a general framework for analyzing the delay and fairness properties of variable-rate and hierarchical servers. We demonstrate, both empirically and analytically, that having a PFQ algorithm with a low WFI value is a prerequisite for constructing H-PFQ servers that provide tight delay bounds. Finally, we propose a new PFQ algorithm called WF^2Q+ that is the first to have the following three properties: (a) providing the tightest delay bound among all PFQ algorithms, (b) having the smallest WFI among all PFQ algorithms, and (c) having a relatively low complexity of O(log N). The resulted H-WF^2Q+ provides similar delay bounds and bandwidth distribution to those provided by the idealized H-GPS server, and is the first in the literature that provides both provably tight delay bounds for real-time sessions and the full semantics of hierarchical link-sharing service.

9 Acknowledgement

We would like to thank Donpaul Stephens and Ion Stoica for helping with the simulations, Harrick Vin for stimulating discussions on SFQ, and anonymous reviewers for their insightful comments.

References

[1] 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.

[2] D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. In *Proceedings of ACM SIGCOMM'92*, pages 14–26, Baltimore, Maryland, August 1992.

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

[4] J. Davin and A. Heybey. A simulation study of fair queueing and policy enforcement. *Computer Communication Review*, 20(5):23–29, October 1990.

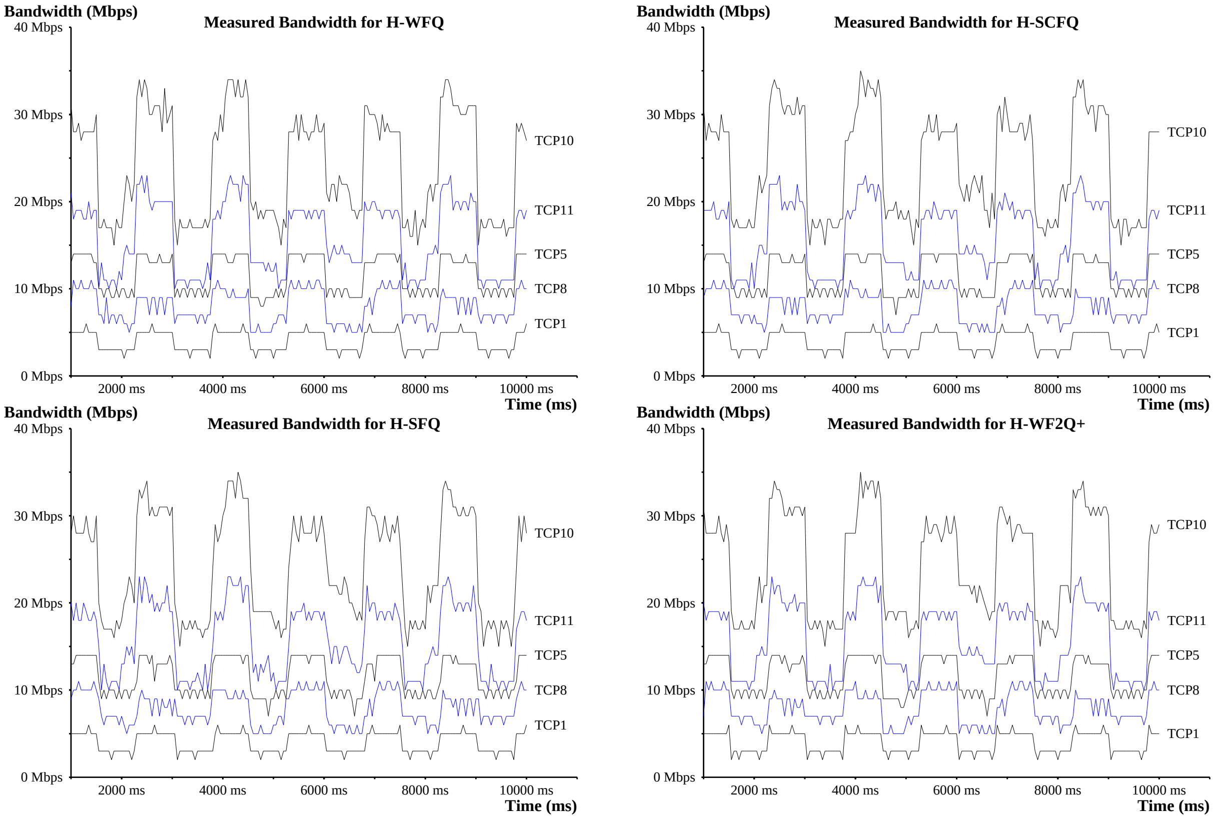


Figure 7: Hierarchical Link-Sharing for H-PFQ's

[5] 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.

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

[8] 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.

[9] S. Keshav. A control-theoretic approach to flow control. In *Proceedings of ACM SIGCOMM'91*, pages 3–15, Zurich, Switzerland, September 1991.

[10] P. McKenney. Stochastic fair queueing. In *Proceedings of IEEE INFOCOM'90*, San Francisco, CA, June 1990.

[11] O. Ndiaye. An efficient implementation of a hierarchical weighted fair queue packet scheduler. Master's thesis, Massachusetts Institute of Technology, May 1994.

[12] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control - the single node case. *ACM/IEEE Transactions on Networking*, 1(3):344–357, June 1993.

[13] S. Suri and G. Varghese and G. Chandranmenon. Leap Forward Virtual Clock. In *Proceedings of INFOCOM 97*, Kobe, Japan, April 1997.

[14] S. Shenker. Making greed work in networks: A game theoretical analysis of switch service disciplines. In *Proceedings of ACM SIGCOMM'94*, pages 47–57, London, UK, August 1994.

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

[16] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *Proceedings of SIGCOMM'95*, pages 231–243, Boston, MA, September 1995.

[17] D. Stilliadis and A. Verma. Design and analysis of frame-based fair queueing: A new traffic scheduling algorithm for packet-switched networks. In *Proceedings of ACM SIGMETRICS'96*, May 1996.

[18] D. Stilliadis and A. Verma. Latency-rate servers: A general model for analysis of traffic scheduling algorithms. In *Proceedings of IEEE INFOCOM'96*, pages 111–119, San Francisco, CA, March 1996.

[19] 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.

[20] H. Zhang and S. Keshav. Comparison of rate-based service disciplines. In *Proceedings of ACM SIGCOMM'91*, pages 113–122, Zurich, Switzerland, September 1991.

A Proof of Theorem 1

Let d_i^k be the time packet p_i^k departs the H-PFQ and $[t_1, d_i^k]$ be a time period that session i is continuously backlogged. It immediately follows that (a) d_i^k is also the time that packet p_i^k departs from server node $p^{h+1}(i)$ and (b) the logical queue at node $p^h(i)$ is continuously backlogged during $[t_1, d_i^k]$ with respect to server node $p^{h+1}(i)$, h=0, $\cdots$, H-1.

Since server node $p^{h+1}(i)$ is worst-case fair with the logical queue at node $p^h(i)$, the following holds for

$$W_{p^h(i)}(t_1, d_i^k) \geq \frac{\phi_{p^h(i)}}{\phi_{p^{h+1}(i)}} W_{p^{h+1}(i)}(t_1, d_i^k) - \alpha_{p^h(i)} \tag{31}$$

where $W_{p^h(i)}(t_1, d_i^k)$ is the amount of service received by node $p^h(i)$ in $[t_1, d_i^k]$. Multiplying $\frac{\phi_i}{\phi_{p^h(i)}}$ at both sides of (31), we have

$$\frac{\phi_i}{\phi_{p^h(i)}} W_{p^h(i)}(t_1, d_i^k) \geq \frac{\phi_i}{\phi_{p^{h+1}(i)}} W_{p^{h+1}(i)}(t_1, d_i^k) - \frac{\phi_i}{\phi_{p^h(i)}} \alpha_{p^h(i)} \tag{32}$$

Summing (32) for h=0, $\cdots$, H-1 and eliminating common terms on both sides, we have

$$W_i(t_1, d_i^k) \geq \frac{\phi_i}{\phi_{p^H(i)}} W(t_1, d_i^k) - \sum_{h=0}^{H-1} \frac{\phi_i}{\phi_{p^h(i)}} \alpha_{p^h(i)} \tag{33}$$

Q.E.D.

B Proof of Theorem 2

Consider the k^{th} packet of session i. Let a_i^k and d_i^k be its arrival and departure times respectively. Based on the definition of SBI, for d_i^k, there exists an instant t_1 within the node $p(i)$ busy period that includes also d_i^k, where $t_1 < d_i^k$ $Q_i(t_1^-) = 0$, and $Q_i(t_1) \neq 0$ holds, such that

$$W_i(t_1, d_i^k) \geq \frac{\phi_i}{\phi_{p(i)}} W_{p(i)}(t_1, d_i^k) - (r_i D_i - \sigma_i) \tag{34}$$

Since both t_1 and d_i^k are in the same server busy period of node $p(i)$, the logical queue at node $p^h(i)$ is continuously backlogged with respect to server node $p^{h+1}(i)$, h=1, $\cdots$, H-1. Also, server node $p^{h+1}(i)$ is worst-case fair with the logical queue at node $p^h(i)$, therefore (31) holds. Multiplying $\frac{\phi_{p(i)}}{\phi_{p^h(i)}}$ at both sides of (31), we have

$$\frac{\phi_{p(i)}}{\phi_{p^h(i)}} W_{p^h(i)}(t_1, d_i^k) \geq \frac{\phi_{p(i)}}{\phi_{p^{h+1}(i)}} W_{p^{h+1}(i)}(t_1, d_i^k) - \frac{\phi_{p(i)}}{\phi_{p^h(i)}} \alpha_{p^h(i)} \tag{35}$$

Summing (35) for h=1, $\cdots$, H-1, and eliminating common terms on both sides, we have:

$$W_{p(i)}(t_1, d_i^k) \geq \frac{\phi_{p(i)}}{\phi_{p^H(i)}} W_{p^H(i)}(t_1, d_i^k) - \sum_{h=1}^{H-1} \frac{\phi_{p(i)}}{\phi_{p^h(i)}} \alpha_{p^h(i)} \tag{36}$$

Combining (34) and (36), we have

$$W_i(t_1, d_i^k) \geq r_i(d_i^k - t_1) - \sum_{h=1}^{H-1} \frac{\phi_i}{\phi_{p^h(i)}} \alpha_{p^h(i)} - r_i D_i + \sigma_i \tag{37}$$

Since session i queue is FIFO and leaky bucket constrained, (14) and (15) holds. Combining them with (37), we have

$$\sigma_i + r_i(a_i^k - t_1) \geq r_i(d_i^k - t_1) - \sum_{h=1}^{H-1} \frac{\phi_i}{\phi_{p^h(i)}} \alpha_{p^h(i)} - r_i D_i + \sigma_i \tag{38}$$

Rearranging terms and dividing both sides by r_i

$$d_i^k - a_i^k \leq D_i + \sum_{h=1}^{H-1} \frac{\alpha_{p^h(i)}}{r_{p^h(i)}} \tag{39}$$

Q.E.D.