Hierarchical Packet Fair Queueing Algorithms

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

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

Abstract

Hierarchical Packet Fair Queueing (H-PFQ) algorithms have the potential to simultaneously support guaranteed real-time service, rate-adaptive best-effort, and controlled link-sharing service. In this paper, we design practical H-PFQ algorithms by using one-level Packet Fair Queueing (PFQ) servers as basic building blocks, and develop techniques to analyze delay and fairness properties of the resulted H-PFQ servers. We demonstrate that, in order to provide tight delay bounds in a H-PFQ server, it is essential for the one-level PFQ servers to have small Worst-case Fair Indices (WFI). We propose a new one-level PFQ algorithm called WF^{2}Q+ 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 implementation complexity of O(log N). We show that practical H-PFQ algorithms can be implemented by using WF^{2}Q+ as the basic building block and prove that the resulting H-WF^{2}Q+ algorithms provide similar delay bounds and bandwidth distribution as those provided by a H-GPS server. Simulation experiments are presented to evaluate the proposed algorithm.

1 Introduction

Future integrated services networks [3, 7, 19] will support multiple service classes that include real-time service, best-effort service, and others. In addition, they will need to support link-sharing [8], which allows resource sharing among applications that require different network services but belong to the same administrative class.

In packet-switched networks, packets from different sessions belonging to different service classes and administrative classes interact with each other when they are multiplexed at the same output link of a switch. The scheduling algorithms at the switching nodes play a critical role in controlling the interactions among different traffic streams, different service classes, and different link sharing classes. Rather than having separate mechanisms to control each of these interactions, it is desirable to have one common mechanism.

The hypothetical Hierarchical Generalized Processor Sharing (H-GPS) algorithm 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 multiple one-level GPS servers. With a one-level GPS, there are multiple queues and a pre-specified service share for each queue. During any time interval when there are backlogged queues the server services all backlogged queues simultaneously in proportion to their corresponding service shares. With H-GPS, the queue at each intermediate 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. We will give the formal definition in Section 2.

Now consider the example shown in Figure 1 (a). There are 11 agencies or organizations sharing the same 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%. This maps nicely to the H-GPS system as illustrated in Figure 1 (b).

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 [14]; (2) bandwidth is fairly distributed to competing sessions [6] and (3) the sources can accurately estimate the available bandwidth to them in a distributed fashion [11]. The first property forms the basis for supporting real-time traffic [3] and the third property enables robust and distributed end-to-end traffic management algorithms for best-effort traffic [11, 15]. Having a hierarchical GPS affects only the distribution of excess bandwidth unused by each subclass, but not the first or third properties. 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.

The above example shows that H-GPS is an ideal mechanism to support integrated services networks. Since H-GPS is defined in a hypothetical fluid system, packet algorithms that approximate H-GPS need to be designed. A number of one-level Packet Fair Queueing (PFQ) algorithms have been proposed to approximate the one-level GPS algorithm [9, 14]. To reduce the implementation complexity, they all use the notion of a 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 virtual time function does not apply to 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 resulting 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

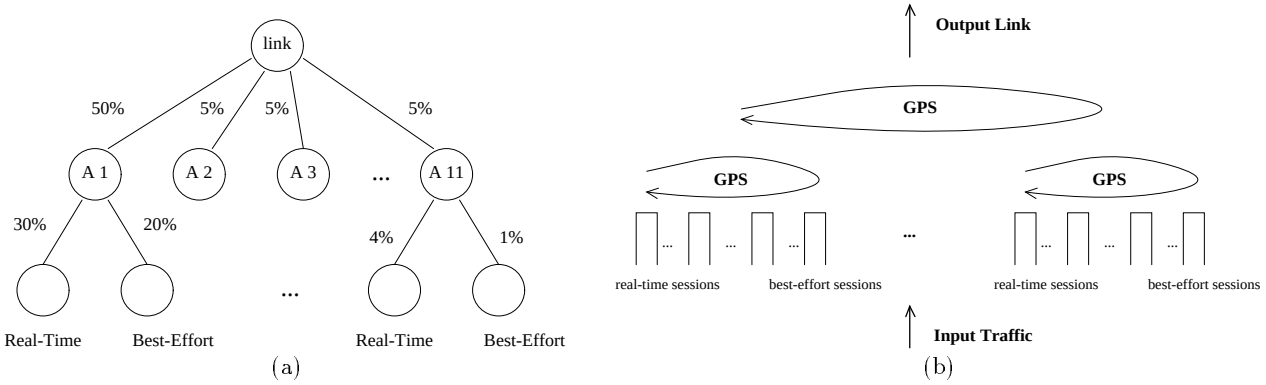


Figure 1: A Link Sharing Example

a H-GPS server; and (3) a relatively low complexity.

To construct such a H-PFQ server, the one-level server needs to have at least 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 have (c) a small Worst-case Fair Index (WFI) as defined in [2]. Most of the previously proposed PFQ algorithms including the popular Weighted Fair Queueing (WFQ) [6, 14] and Self-Clocked Fair Queueing (SCFQ) [9] do not have small WFI's. In fact, they both 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 that are made of WFQ or SCFQ are much larger than those provided by H-GPS. The only exception is Worst-case Fair Weighted Fair Queueing or WF^{2}Q, which is proven to have the smallest WFI among all PFQ algorithms [2]. However, WF^{2}Q uses a virtual time function with a high complexity.

In this paper, we propose a new algorithm that maintains all the important properties of WF^{2}Q, but has a lower complexity than WF^{2}Q. We call the new algorithm WF^{2}Q+ . We show that practical H-PFQ algorithms can be implemented by using WF^{2}Q+ as the basic building block and prove that the resulting H-WF^{2}Q+ algorithms preserve both the bounded-delay and fairness properties of H-GPS.

The rest of the paper is organized as follows. In Section 2, we review issues of designing packet approximation algorithms for fluid algorithms in the context of both one-level GPS and H-GPS systems. In Section 3, we analyze the delay and fairness property of H-PFQ servers, and show that having PFQ's with small WFI values are pre-requisite for providing tight delay bounds in a H-PFQ server. We discuss limitations of existing PFQ algorithms and present a new algorithm WF^{2}Q+ that overcome these limitations. In Section 4, we describe the detailed algorithm to construct a H-PFQ server by using PFQ servers as basic building blocks. In Section 5, we present simulation experiments to illustrate the properties of H-WF^{2}Q+ .

2 Background: Fluid Systems and Packet Systems

Throughout the paper, we discuss two types of systems: fluids system in which the traffic is infinitely divisible and the server can serve traffic streams 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 their 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 Approximating GPS with Packet Algorithms

A one-level GPS server with N queues is characterized by N positive real numbers, $\phi_1, \phi_2, \cdots, \phi_N$. During any time interval when there are exactly M non-empty queues, the server serves the M packets at the head of the queues simultaneously, in proportion to their service shares. Formally, let $W_i(t_1, t_2)$ be the amount of session i traffic served in the interval $[t_1, t_2]$. Then a work-conserving GPS server is defined as one for which

$$\frac{W_i(t_1, t_2)}{W_j(t_1, t_2)} \geq \frac{\phi_i}{\phi_j} \qquad j = 1, 2, \cdots, N \tag{1}$$

holds for any queue i that is backlogged throughout the interval $[t_1, t_2]$ [14]. It immediately follows from the definition that

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

holds for any two queues i and j that are backlogged throughout the interval $[t_1, t_2]$, i.e., the server distributes bandwidth fairly to all backlogged queues in proportion to their service shares.

For ease of discussion and without losing generality, we restrict the assignment of ϕ's so that $\sum_{i=1}^{N} \phi_i = 1$ holds. Let $r_i = \phi_i r$ where r is the server rate, it can be shown that

$$W_i(t_1, t_2) \geq r_i(t_2 - t_1) \tag{3}$$

holds for any interval $[t_1, t_2]$ during which queue i is continuously backlogged, i.e., queue i gets a minimum service rate of r_i during any period when it is backlogged *regardless of the behavior of other sessions*. 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 [14].

A good packet approximation algorithm of GPS would be one that serves packets in increasing order of their finish times in the fluid system [6, 14]. However, when the packet

system is ready to choose the next packet to transmit, it is possible that the next packet to depart under the fluid 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) [6], 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.

A practical implementation of WFQ can be designed based on the following important property of the one-level GPS system [14]:

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. An implementation based on the notion of a virtual time function is proposed in [6, 14]. During any system busy period $[t_1, t_2]$, $V_{GPS}(\cdot)$ is defined as follows:

$$V_{GPS}(t_1) = 0 \quad (4)$$

$$\frac{\partial V_{GPS}(\tau)}{\partial \tau} = \frac{1}{\sum_{i \in B_{GPS}(\tau)} \phi_i} \quad \forall t_1 \le \tau \le t_2 \quad (5)$$

where $B_{GPS}(\tau)$ is the set of backlogged queues at time τ. In GPS, if connection j is backlogged at time τ, it receives a service rate of $\frac{\partial V_{GPS}(\tau)}{\partial \tau}\phi_j r$, where r is the link speed. Therefore, $V_{GPS}(\cdot)$ can be interpreted as the marginal rate at which backlogged queues receive service in GPS. For the k^{th} packet on session i, its virtual start and finish times S_i^k, F_i^k are defined as:

$$S_i^k = max\{F_i^{k-1}, V(a_i^k)\} \quad (6)$$

$$F_i^k = S_i^k + \frac{L_i^k}{r_i} \quad (7)$$

while $F_i^0 = 0$, and a_i^k and L_i^k are the arrival time and the length of the packet respectively. Based on Property 1, for all the packets that are present in the WFQ 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 as follows: when the server selects the next packet for service at time τ, it always picks the packet with the smallest virtual finish time. We call such a policy "Smallest virtual Finish time First" (SFF) policy.

This implementation has two advantages: (a) the virtual finish time of a packet can be calculated at the packet arrival time, therefore there is no need to calculate packet finish times in GPS system each time the server picks the next packet to transmit; (b) a priority queue mechanism based on packet virtual finish times can be used to maintain the relative packet finish order in the GPS system. The complexity of each insertion or deletion operation for the queue is O(log N).

Besides maintaining a priority queue, there is also the cost of maintaining the virtual time function $V_{GPS}(.)$ with this implementation. While the average complexity of computing $V_{GPS}(.)$ is O(1), the worst complexity can be O(N) [9]. It is possible to have other Packet Fair Queueing algorithms based on virtual time functions with lower worst-case complexity. For example, in [9], Golestani proposed a virtual time function V_{SCFQ} with a worst-case complexity of O(1). 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 Hierarchical Generalized Processor Sharing (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, then a work-conserving H-GPS server is defined as one for which

$$\frac{W_n(t_1, t_2)}{W_m(t_1, t_2)} \ge \frac{\phi_n}{\phi_m} \quad (8)$$

holds if (a) node n is backlogged throughout the interval $[t_1, t_2]$, and (b) m, n are sibling nodes that share the same parent node. It immediately follows that

$$\frac{W_m(t_1, t_2)}{\phi_m} = \frac{W_n(t_1, t_2)}{\phi_n} \quad (9)$$

holds for any two sibling nodes m, n that are backlogged throughout the interval $[t_1, t_2]$.

Comparing (8) and (9) with (1) and (2), we can see that they are very similar. The major difference is that while these relationships hold for any two queues in GPS, they hold only for sibling nodes in H-GPS – notice that packet queues are associated with only leaf nodes in H-GPS. In H-GPS, the bandwidth is not always distributed to all queues in proportion to their service shares as in GPS. Instead, each node receives bandwidth from its parent node and in turn distributes it to its its children in proportion to the relative service shares among themselves. Therefore, if two queues have the same $\phi's$ and are both backlogged, it is possible that they receive different amount of service due to the different relative shares between their ancestor nodes.

As in the case of one-level GPS, we assume the sum of the $\phi_i's$ of all session, i.e., $\sum_{i \in leaf(R)} \phi_i = 1$. In addition, we assume $\sum_{m \in child(n)} \phi_m = \phi_n$. It immediately follows that $\phi_R = 1$. Let $r_n = \phi_n r$, it can be easily shown that (3) holds also for H-GPS. Therefore, H-GPS can provides the same minimum bandwidth and delay bound guarantees for each session as GPS. As discussed above, the major difference between GPS and H-GPS is the distribution of the excess bandwidth when a session cannot fully utilize its guaranteed bandwidth. While GPS distributes the excess bandwidth fairly among all backlogged queues, H-GPS prioritizes the distribution according to the hierarchy. Sessions that share smaller subtrees with the session of excess bandwidth have higher priorities.

As in the case of GPS, H-GPS needs to be approximated by a packet algorithm. While it is possible to design practical implementations of WFQ based on virtual time functions, this is not the case with H-GPS. The main reason is

that Property 1 does not hold for H-GPS, i.e., *the relative order of packet finish time in the fluid system 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 children leaf nodes A1 and A2 with service shares of 0.75 and 0.05 respectively. 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, which will become 21, 41, 61, *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, the ratio between the service received by any two sessions during any period in which they are both backlogged is a constant regardless of the 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.

The fact that H-GPS does not have the relative-packet-order-invariant property means that the packet approximation algorithm defined above cannot be implemented using a virtual time function. Therefore, when the packet server is picking the next packet to transmit, if the set of backlogged sessions has changed since last picking, it needs to recompute the H-GPS finish time for each packet at head of a queue and perform a minimization operation. Since such an implementation has a complexity of $O(N)$, where N is the number of sessions, it is not practical for high speed implementation.

3 PFQ's as Building Blocks for H-PFQ

In the previous section, we have shown that the same approach of approximating GPS by using a virtual time function does not directly apply to H-GPS. In this paper, we propose to implement packet approximation algorithms for H-GPS by using one-level PFQ servers as basic building blockings and organizing them in a hierarchical structure. We call the resulting algorithms Hierarchical Packet Fair Queueing (H-PFQ) algorithms.

In this section, we show that the delay bound provided by a H-PFQ server relates not only to the delay bound provided by PFQ server nodes in the hierarchy, but also to the Worst-case Fair Index (WFI) of the PFQ servers. We propose a new PFQ algorithm called WF^{2}Q that not only provides tight delay bound and low WFI, but also has a relatively low complexity. We demonstrate that practical H-PFQ can be constructed using WF^{2}Q+ .

3.1 Limitation of WFQ

Weighted Fair Queueing (WFQ) [3, 6], also known as Packetized Generalized Processor Sharing [14], is the best-known packet approximation algorithm for GPS. It has been shown

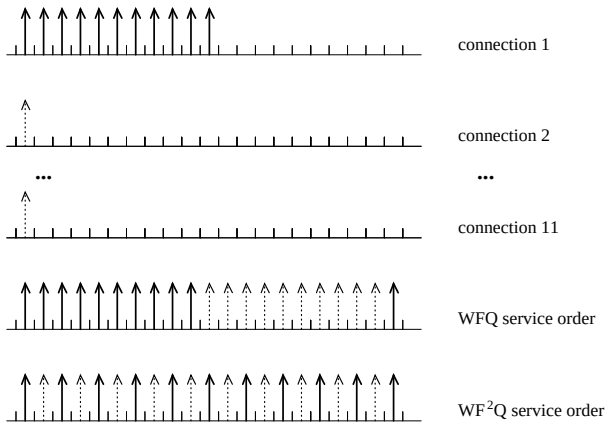


Figure 2: WFQ and WF^{2}Q

that the delay bound provided by WFQ is within one packet transmission time of that provided by GPS [14].

Even though WFQ and GPS have almost identical delay bounds, there could be large discrepancies between the services provided by these two systems. Consider the example illustrated in Fig. 2 where there are 11 sessions sharing the same link. For simplicity, assume all packets have the same size of 1 and the link speed is 1. Also, let the guaranteed rate for session 1 be 0.5, and the guaranteed rate for each of the other 10 sessions be 0.05. In the example, 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 on session 1, one time unit to transmit the 11^{th} packet, and 20 time units to transmit the first packet from another session. Denote the k^{th} packet on session j to be p^k_j, then in the GPS system, the finish time is $2k$ for $p^k_1, k = 1 \ldots 10$, 21 for p^{11}_1, and 20 for $p^1_j, j = 2 \cdots 11$. Under WFQ, since the first 10 packets on session 1 ($p^k_1, k = 1 \ldots 10$) all have GPS finish times smaller than packets on other sessions, the server will transmit 10 packets on session 1 back to back before transmit packets from other sessions. After the burst, the next packet on session 1, p^{11}_1, will have a larger finishing time in the GPS system than the 10 packets at the head of other sessions' queues, therefore, it will not be served until all the other 10 packets are transmitted. This is shown in Fig. 2. In the above 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 transmit provided to each session by WFQ and GPS is a measure of inaccuracy of WFQ in approximating GPS. Therefore, the inaccuracy introduced by WFQ is *not* merely one packet, but $N/2$ packets, where N is the number of sessions sharing the link.

The inaccuracy introduced by WFQ will significantly affect the delay bound in a hierarchical server. 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.

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

In [2], we introduce a metric called Worst-case Fair Index (WFI) to characterize PFQ servers. In this session, we will show that having PFQ servers with small WFIs is a prerequisite to constructing a H-PFQ server that provides tight delay bounds.

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,*

$$d_i^k - a_i^k \leq \frac{Q_i(a_i^k)}{r_i} + \mathcal{A}_{i,s} \tag{10}$$

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 k^{th} packet of session i respectively.

Intuitively, $\mathcal{A}_{i,s}$ represents the maximum time a packet coming to an empty session 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 an empty session 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 be quite large. In fact, it is shown in [2] that WFI for WF^2Q increases linearly as a function of the number of sessions N.

Since the previous definition of WFI applies only to a standalone server, we introduce a general definition of WFI that applies also to a server node in a hierarchical system. A non-root server node differs from a standalone server in two aspects: (a) it receives its service from its parent and therefore does not have a constant service rate; (b) the queues it serves do not have to be FIFO.

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 during any time interval $[t_1, t_2]$ when the queue is continuously backlogged, the following holds*

$$W_i(t_1, t_2) \geq \frac{\phi_i}{\phi_s} W_s(t_1, t_2) - \alpha_{i,s} \tag{11}$$

where $\frac{\phi_i}{\phi_s}$ is the service share guaranteed to queue i by server s.

For a standalone server, $\phi_s = 1$, $W_s(t_1, t_2) = r(t_2 - t_1)$. Therefore, (11) is equivalent to:

$$W_i(t_1, t_2) \geq r_i(t_2 - t_1) - \alpha_{i,s} \tag{12}$$

To see that Definition 2 is equivalent to Definition 1 in the case of a standalone server, we first observe the following property for a FIFO queue i,

$$W_i(a_i^k, d_i^k) = Q_i(a_i^k) \tag{13}$$

Plugging (13) into (12) by letting $t_1 = a_i^k$ and $t_2 = d_i^k$

$$Q_i(a_i^k) \geq r_i(d_i^k - a_i^k) - \alpha_{i,s} \tag{14}$$

Rearranging the terms, we have

$$d_i^k - a_i^k \leq \frac{Q_i(a_i^k)}{r_i} + \frac{\alpha_{i,s}}{r_i} \tag{15}$$

Therefore, the two definitions are equivalent for a standalone server, with $\mathcal{A}_{i,s}$ and $\alpha_{i,s}$ measures WFI in unit of time and bit respectively, and $\alpha_{i,s} = r_i\mathcal{A}_{i,s}$.

Before we proceed to establish the relationship between the delay bound of a H-PFQ server and WFI's of PFQ server nodes, we first give the following definition of guaranteed service burstiness index (SBI) as proposed in [4]. We will show that the guaranteed service burstiness property can be viewed as a generalized bounded delay property that applies not only to a standalone server, but also to a server node in a hierarchical system.

Definition 3 *A server node s is said to guarantee a service burstiness index (SBI) of $\gamma_{i,s}$ to session i if for any time instant t_2 when session i is backlogged, there exists another time instant t_1 within the same server busy period of t_2, where $t_1 < t_2$, $Q_i(t_1^-) = 0$, and $Q_i(t_1) \neq 0$ hold, such that*

$$W_i(t_1, t_2) \geq \frac{\phi_i}{\phi_s} W_s(t_1, t_2) - \gamma_{i,s} \tag{16}$$

where $\frac{\phi_i}{\phi_s}$ is the service share guaranteed to queue i by server s.

While looking similar, worst case fair is a stronger property than bounded service burstiness. In the case of the worst-case fair property, (16) needs to hold for *all intervals* when the session is continuously backlogged. In the case of the guaranteed service burstiness property, for each time instant t_2 when the session is backlogged, (11) needs to hold for only *one* interval that ends at t_2 and starts at the beginning of a session i's backlogged period. Notice that t_1 and t_2 need not to belong to the same session i backlogged period. By letting t_1 to be the start of the backlogged period that includes t_2, 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 $N * Pmax$. Therefore, worst-case fair is a stronger property than bounded service burstiness with the same index value.

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 $[t_1, t_2]$

$$A_i(t_1, t_2) \leq \sigma_i + \rho_i(t_2 - t_1) \tag{17}$$

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 guarantees a SBI of $\gamma_{i,s}$ to the session, it can guarantees a delay bound of $\frac{\sigma_i + \gamma_{i,s}}{r_i}$*

Proof. For a standalone server, $\phi_s = 1$ and $W_s(t_1, t_2) = r(t_2 - t_1)$ hold. Let a_i^k and d_i^k be the arrival and departure

time of the k^{th} packet of session i respectively. Since the server guarantees a SBI to session i, there exists a t_1, where $t_1 < d_i^k$, $Q_i(t_1^-) = 0$, and $Q_i(t_1) \neq 0$ hold, such that

$$\begin{aligned} W_i(t_1, d_i^k) &\geq \frac{\phi_i}{\phi_s} W_s(t_1, d_i^k) - \gamma_{i,s} \\ &= r_i(d_i^k - t_1) - \gamma_{i,s} \end{aligned} \quad (18)$$

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

$$W_i(t_1, d_i^k) = A_i(t_1, a_i^k) \quad (19)$$

In addition, session i is leaky bucket constrained, thus

$$A_i(t_1, a_i^k) \leq \sigma_i + r_i(a_i^k - t_1) \quad (20)$$

Combining (18), (19), and (20), we have

$$\sigma_i + r_i(a_i^k - t_1) \geq r_i(d_i^k - t_1) - \gamma_{i,s} \quad (21)$$

Rearranging the terms and dividing both sides by r_i, we have

$$d_i^k - a_i^k \leq \frac{\sigma_i + \gamma_{i,s}}{r_i} \quad (22)$$

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 a SBI of $r_i D_i - \sigma_i$ to the session [4]. 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 server nodes in a hierarchy, it can be used as a 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 $\frac{\sigma_i}{r_i} + \frac{Pmax}{r}$, the delay bound based on the WFI can be $\frac{\sigma_i}{r_i} + N\frac{Pmax}{r}$, 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 the 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 the 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 and delay bound for an H-PFQ server. For a session i with H ancestors in a 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 $p^0(i) = i$ and $p^1(i) = p(i)$ hold. It follows directly that $p^H(i)$ is the root node R.

Theorem 1 *For a session i with H ancestors in a H-PFQ server, the following holds*

$$\alpha_{i,H-PFQ} = \sum_{h=0}^{H-1} \frac{\phi_i}{\phi_{p^h(i)}} \alpha_{p^h(i)} \quad (23)$$

where $\alpha_{i,H-PFQ}$ is the B-WFI for session i queue with the H-PFQ server, $\alpha_{p^h(i)}$ 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 for a H-PFQ server is the sum of WFI's weighted by the guaranteed rate for all the PFQ server nodes along the path from the leaf node to the root node.

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

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

$$\frac{\sigma_i}{r_i} + \sum_{h=0}^{H-1} \frac{\alpha_{p^h(i)}}{r_{p^h(i)}} \quad (24)$$

While Corollary 1 gives the delay bound for a leaky bucket constrained session in a 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 for a H-PFQ server.

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

$$D_i + \sum_{h=1}^{H-1} \frac{\alpha_{p^h(i)}}{r_{p^h(i)}} \quad (25)$$

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 weighted by the guaranteed rate for all the other PFQ server nodes along the path from the leaf node to the root node. To achieve tight delay bounds in a H-PFQ server, the WFI's for the intermediate and root server nodes should be small.

3.3 WF^{2}Q

We have shown that to achieve tight delay bounds in a H-PFQ server, it is important for the PFQ server nodes to have small WFI's.

Most of the previously proposed PFQ algorithms have large WFI's. In [2], we present an algorithm called Worst-case Fair Weighted Fair Queueing (WF^{2}Q). 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. 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 less than or equal to the current (virtual) time.

We call the policy used by WF^{2}Q "Smallest Eligible virtual Finish time First" (SEFF) policy If we consider again the example in Section 3.1, 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 FFQ system. Among them, p_1^1 has the smallest finish time in FFQ, 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 FFQ system, thus are eligible, and one of them will be transmitted. p_2^1 as the next packet for service. At time 3, p_1^2 becomes eligible and has the smallest finish time among all packets, thus it will start service next. The service order for all packets under WF^{2}Q is shown as the last time line in Fig. 2. During any interval in the example, 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 [2].

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

$$\alpha_{i,WF^2Q} = L_{i,max} + (L_{max} - L_{i,max})\frac{r_i}{r} \qquad (26)$$

(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 for 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 transmission time, this means that WF^{2}Q is an optimal packet policy with respect to the worst-case fair property. In addition, since the maximum 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.

3.4 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 key aspect of WF^{2}Q+ is the use of a new 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 [9, 18], they all result in PFQ algorithms with large WFI's. The unique advantage of $V_{WF^2Q+}(\cdot)$ is that the resulting 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 follows,

$$V_{WF^2Q+}(t+\tau) = \max(V_{WF^2Q+}(t) + \tau,\ min_{i \in \hat{B}(t)}(S_i^{h_i(t)})) \qquad (27)$$

where $\hat{B}(t)$ 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, it 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+}. As discussed in [1], having a virtual time function with the minimum slope property is a necessary and sufficient condition 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, satisfies this requirement 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 function used by SCFQ [9] may have a slope of 0 during certain periods, and therefore the delay bounds provided by the resulting SCFQ algorithm are much larger than those provided by WFQ and WF^{2}Q [10]. The second important property of $V_{WF^2Q+}(\cdot)$, as provided by the max over min operation in (27), 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 resulting 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 less than the current virtual time. This guarantees the resulting SEFF policy to be work-conserving as only packets with virtual start time less than or equal to the current virtual time are eligible for transmission.

To simplify the implementation, we also modify the definition of virtual start and finish times as defined in (6) and (7). With the old definition, 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

$$S_i = \begin{cases} F_i & if Q_i(a_i^k -) \neq 0 \\ max(F_i, V(a_i^k)) & if Q_i(a_i^k -) = 0 \end{cases} \qquad (28)$$

$$F_i = S_i + \frac{L_i^k}{r_i} \qquad (29)$$

where $Q_i(a_i^k -)$ is the queue size of session i just before time a_i^k. With this definition, the 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) computation of the virtual time function; and (b) maintaining the set of eligible sessions sorted by virtual finish times. As shown in [1], both can be accomplished with O(log N) complexity.

The following theorem is proven in [1]

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

$$\alpha_{i,WF^2Q+} = L_{i,max} + (L_{max} - L_{i,max})\frac{r_i}{r} \qquad (30)$$

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

Therefore, WF^{2}Q+ has the same worst-case fairness and bounded delay properties as WF^{2}Q, but uses a 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 $H-WF^2Q+$ 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*

$$\frac{\sigma_i}{r_i} + \sum_{h=0}^{H-1} \frac{L_{max}}{r_{p^h(i)}} \qquad (31)$$

4 Implementation of H-PFQ

In this section, we first discuss the general issues that are involved in converting a standalone PFQ server into a PFQ server node within a hierarchical structure. We then present the detailed algorithm of H-WF^{2}Q+ .

4.1 Standalone Server and Server Node in a Hierarchy

As discussed in Section 3.2, a PFQ server node in a hierarchy differs from a standalone PFQ server in two aspects: (a) it receives its service from its parent and therefore does not have a constant service rate; (b) the queues it serves do not have to be FIFO. We need to address the following two issues when we transform a standalone PFQ into a server node (1) the meaning of "real time" in the computation of virtual time function; and (2) the determination of the head of non-FIFO logical queue.

The virtual time function is a measure of the normalized service that should be provide to each session. For a standalone server that operates at a fix rate, the length of the server busy period is a good reference against which the per session normalized service should be compared. As mentioned in Section 3.4, in order for a standalone PFQ algorithm to provide delay bounds within one packet transmission time of those provided by FFQ, the virtual time function should have the "minimum slope property". If we re-examine the computation of $V_{WF^2Q+}(\cdot)$ as defined in (27), we notice that one element in the formula is τ, which is the time elapsed since the last computation of the virtual time. With τ in the formula, we ensure that the slope of $V_{WF^2Q+}(\cdot)$ is never less than 1. This computation applies only to a server with a constant service rate. For a server that does not have a constant service rate, the total elapsed time no longer provides the same reference point against which we can measure system work.

To address this issue, we define a more general notion of Reference Time that can measure the system work in both constant-rate and non-constant-rate servers. The Reference Time T_n for a server node n in a H-PFQ server is defined to be $T_n(t) = \frac{W_n(0,t)}{r_n}$, where $W_n(0,t)$ is the total amount of bits that have been serviced by node n during period [0,t]. We now show that the Reference Time is equivalent to the real time for the constant-rate root server. For the root node R, $\phi_R = 1$ holds. For any period $[t_1, t_2]$, during which the H-PFQ server is continuously busy, we have:

$$T_R(t_2) - T_R(t_2) = \frac{(t_2 - t_1)\cdot r}{1 \cdot r} = t_2 - t_1 \qquad (32)$$

Therefore, during any system busy period, T_R has the same evolution as the real time t.

The second difference between a server node in a hierarchy and a standalone server 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.

4.2 Pseudocode

To implement H-WF^{2}Q+ , we have a tree representation of the hierarchy. The root node represents the physical link and the leaf node represents a physical queue. Each non-root node n is connected to its parent $p(n)$ by a logical queue Q_n. For the parent node to implement the WF^{2}Q+ 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$(i, Packet)$
1 ENQUEUE$(\widehat{Q}_i, Packet)$
2 **if** $Q_i(t) \neq \emptyset$
3 **then return**
4 $Q_i(t) \leftarrow Packet$
5 $n \leftarrow p(i)$
6 $s_i(t) \leftarrow \max(f_i(t), V_n(T_n(t)))$
7 $f_i(t) \leftarrow s_i(t) + \frac{L_i(t)}{r_i}$
8 **if** $Busy_n = FALSE$
9 **then** RESTART-NODE(n)

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)
1 Pick $m \in E_n(t)$ *s.t.* $f_m(t) = \min_{l \in E_n(t)}(f_l(t))$
2 **if** $m \neq \emptyset$
3 **then**
4 $ActiveChild_n \leftarrow m$
5 $q \leftarrow p(n)$

$T_n(t)$	the *Reference Time* for server n as a function of real time t.
$V_n(T_n(.))$	the *Virtual Time* for server n as a function of its *Reference Time*.
r_n	guaranteed rate for server n.
$s_n(t)$	the virtual starting time of the packet $Q_n(t)$
$f_n(t)$	the virtual finishing time of the packet $Q_n(t)$
$L_n(t)$	the length of the packet $Q_n(t)$
$Busy_n(t)$	a boolean function indicating whether or not node n is transmitting a packet at time t
$p(n)$	the index of the node that is the parent of node n
Q_n	the logical queue for node n
$Q_n(t)$	the packet in the logical queue for node n at time t
$\widehat{Q}_i$	the real queue for the leaf node i
$\widehat{Q}_i(t)$	the packet at the head of the queue $\widehat{Q}_i$ at time t
$ActiveChild_n(t)$	the index of the node $m \in$ children(n) for which $Q_m(t) = Q_n(t)$
C_n	the set composed of the indicies of all nodes which are children of node n.
$\widehat{C}_n(t)$	the set composed of the indices of all nodes $m \in C_n$ s.t. $Q_m(t) \neq \emptyset$
$Smin_n(t)$	$Smin_n(t) = \min_{m \in \widehat{C}_n(t)}(s_m(t))$
$E_n(t)$	$\forall m \in \widehat{C}_n(t^+) \mid s_m(t) \leq \max(V_n(T_n(t)), Smin_n(t^+))$

Table 1: Notations used in the section

```
6          Q_n(t) ← Q_m(t))
7          if Busy_n(t) = TRUE
8            then s_n(t) ← f_n(t)
9            else  s_n(t) ← max(f_n(t), V_q(T_q(t)))
10         f_n(t) ← s_n(t) + L_n(t)/r_n
11         Busy_n ← TRUE
12         V_n(T_n(t)) ← max(V_n(T_n(t)) Smin_n(T)) + L_n(t)/r_n
13         T_n(t) ← T_n(t) + L_n(t)/r_n
14     else
15         ActiveChild_n ← ∅
16         Busy_n ← FALSE
17   if (n ≠ R) and (Q_q(t) ← ∅)
18     then RESTART-NODE(q)
19   if (n = R) and (Q_R(t) ≠ ∅)
20     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 $s_n(t) = \max(f_n(t), V_q(T_q(t)))$. If the node is not idle and has a packet to transmit, the new start time will be set to the current finish time. If the node is not idle but has no more packets to send, busy flag will be cleared to indicate that the node is now idle. If the current node is the root node and there is a packet in the queue, the packet will be transmitted over the link. If the current node is not the root node, and its parent node does not have a packet in its queue Q_q, the node will restart its parent node.

```
RESET-PATH(n)
1   Q_n(t) ← ∅
2   if Leaf(n) = TRUE
3     then
4          DEQUEUE(Q̂_n)
5          if Q̂_n(t) ≠ ∅
6            then
7                 Q_n(t) ← Q̂_n(t)
8                 s_n(t) ← f_n(t)
9                 f_n(t) ← s_n(t) + L_n(t)/r_n
10         RESTART-NODE(p(n))
11    else
12         m ← ActiveChild_n
13         ActiveChild_n ← ∅
14         RESET-PATH(m)
```

When the link finishes serving 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 tree 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.

5 Simulation Experiments

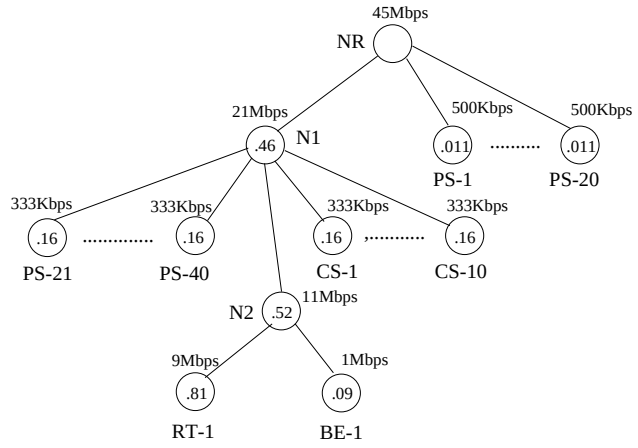
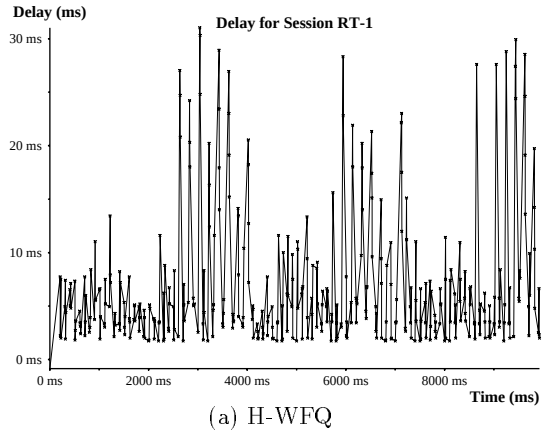
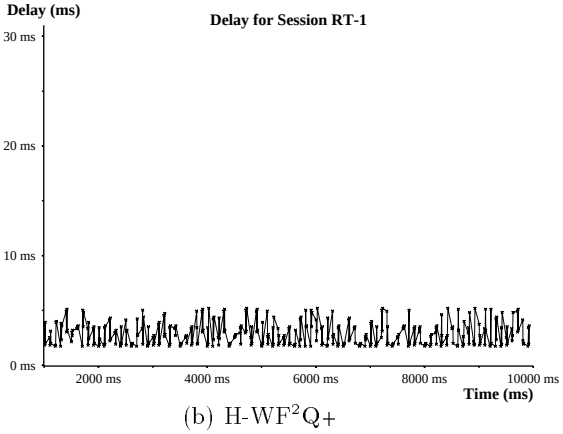


Figure 3: Example 1

In this section, we present simulation experiments to illustrate the bounded delay and hierarchical link-sharing properties of H-WF^{2}Q+ . To demonstrate the tightness of delay

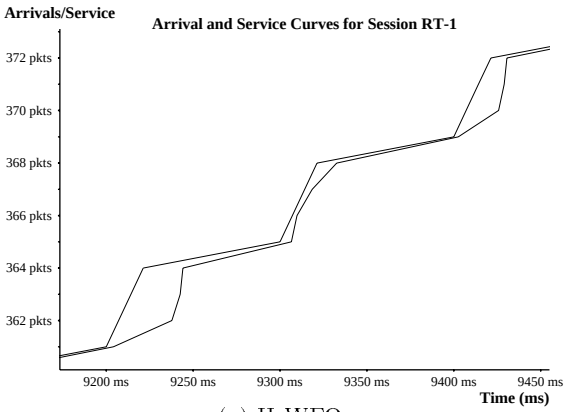


(a) H-WFQ

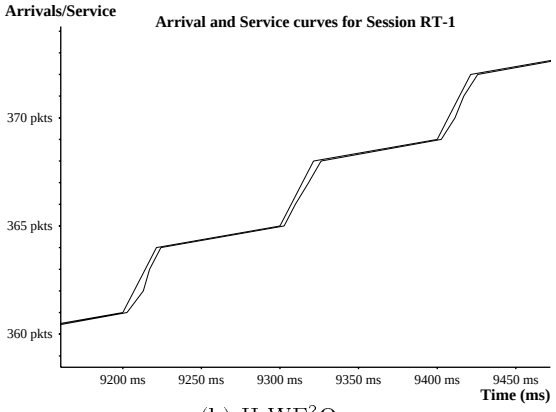


(b) H-WF^{2}Q+

Figure 4: Absolute delay experienced by real-time session under H-WFQ and WF^{2}Q+



(a) H-WFQ



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

Figure 5: Service lag experienced by real-time session under H-WFQ and H-WF^{2}Q+

bounds provided by H-WF^{2}Q+ , we contrast the delay distributions between H-WF^{2}Q+ and H-WFQ servers. These experiments were conducted using a modified version of the NETSIM discrete event simulator from MIT.

5.1 Delay Characteristics

In this section we compare the packet delay distributions for a real-time session under two different H-PFQ servers, H-WF^{2}Q+ and H-WFQ. 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 guaranteed rate as a fraction of it's parent's rate.

The real-time session being measured is the leaf node labeled *RT-1*. It has a guaranteed share of 0.81 from its parent node which translates into a guaranteed rate of 9Mbps. Session *RT-1* is a deterministic on/off source which starts at time $t = 200ms$ with a duty cycle of 25ms on, and 75ms off. Session *RT-1* shares node *N-1* with a sibling *BE-1* . *BE-1* is continuously backlogged and as a result nodes *N-R* , *N-2* and *N-1* are also continuously backlogged. This allows us to see the effects of link-sharing between unconstrained sessions and sessions with delay guarantees. We have two additional types of background traffic, the sessions labeled *PS-n* are poisson traffic sources. while the sessions labeled *CS-n* are constant rate sessions with identical start times and a peak transmission rate equal to their guaranteed rate. The *CS-n* sessions are 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.

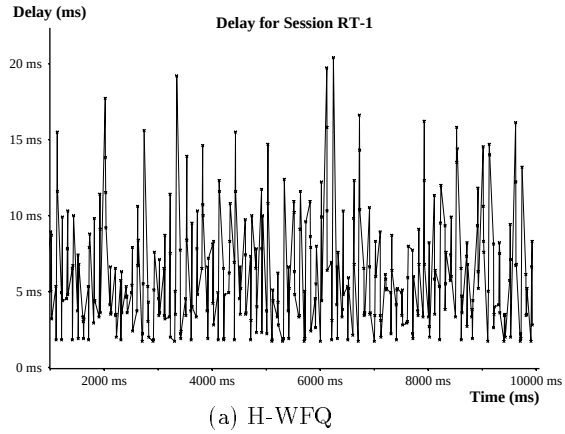
We consider the following three scenarios:

- All sources except *BE-1* are transmitting with average rates their guaranteed rates. Since the sum of the guaranteed rates over all sessions is less than the link speed, only session *BE-1* is continuously backlogged in the system.
- *CS-n* sessions are off, but *PS-n* sources send at an average rate of 1.5 times their guaranteed rate.
- *CS-n* sessions are on and PSn sources continue to send at an average rate of 1.5 times their guaranteed rate.

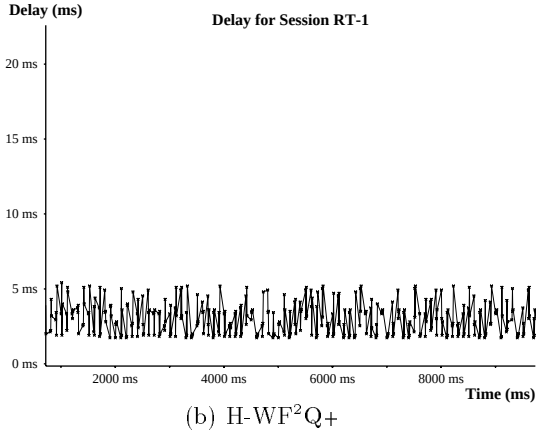
5.1.1 Poisson and Constant Cross Traffic

In Fig. 4(a) we see that under H-WFQ, session *RT-1* experiences large periodic spikes in the observed delay. This is due to the periodic nature of *RT-1* and the *CS-1* , $\cdots$, *CS-10* sessions. *RT-1* becomes active every 100ms, while the *CS* ousrces arrive approximately every 193ms. As a result, there is a cycle of about 3 seconds long.

Fig. 5 shows a close-up of the packet arrivals and service received by session *RT-1* corresponding to the two large

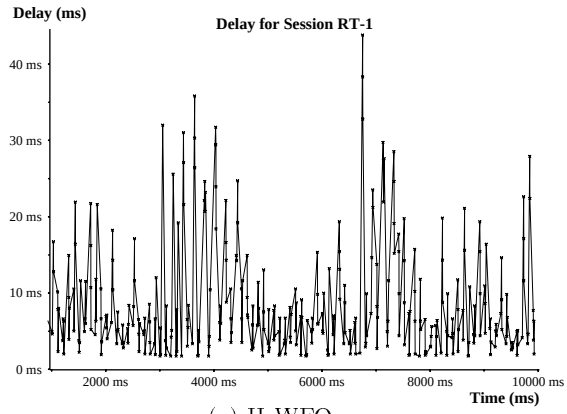


(a) H-WFQ

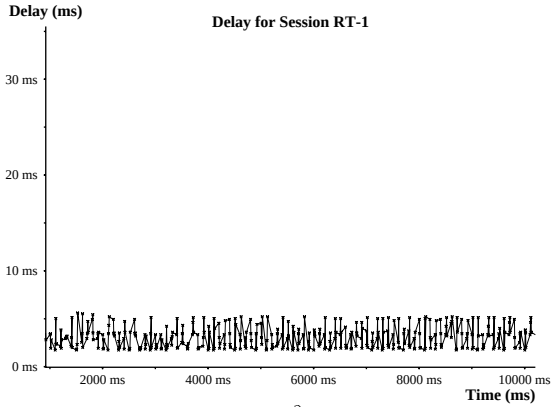


(b) H-WF^{2}Q+

Figure 6: Absolute delay experienced by real-time session with overloaded poisson traffic



(a) H-WFQ



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

Figure 7: Absolute delay experienced by real-time session with overloaded poisson traffic and constant traffic

spikes in Fig 4(a). The upper line is the number of packets arrived at the server at time t, the lower line is the number of packets served by time t. As can be seen, while the two curves track closely with each other under H-WF^{2}Q+ , they can differ by a large amount under H-WFQ.

5.1.2 Overloaded Poisson Cross Traffic

In this section the graphs are for the same network as before with the exception that *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 see in Fig. 6(a) even with purely random initial arrival the maximum delay experienced under H-WFQ is still much greater than that experienced under H-WF^{2}Q+ .

5.1.3 Constant and Overloaded Poisson Cross Traffic

In our next experiment shown in Fig 7, the poisson sessions are still overloaded, and the constant rate sessions are turned back on. Compared to the previous two scenarios, we can see that the worst case delay increases substantially under H-WFQ, but remains almost the same for H-WF^{2}Q+ .

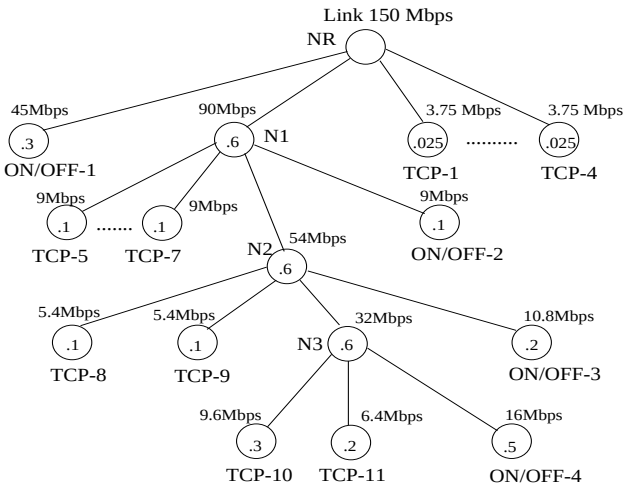
We make the following observations based on the three sets of experiments. For H-WFQ server, (a) correlation among a fraction of traffic streams can have very adverse effects on the delay experienced by other sessions; (b) even when there is no correlation among traffic sources, under overload, the delay experienced by some sessions may be very large; and (c) the effects of any correlated sources are magnified under overload. None of these apply to H-WF^{2}Q+ due to its strong worst-case fairness property.

5.2 Hierarchical Link Sharing

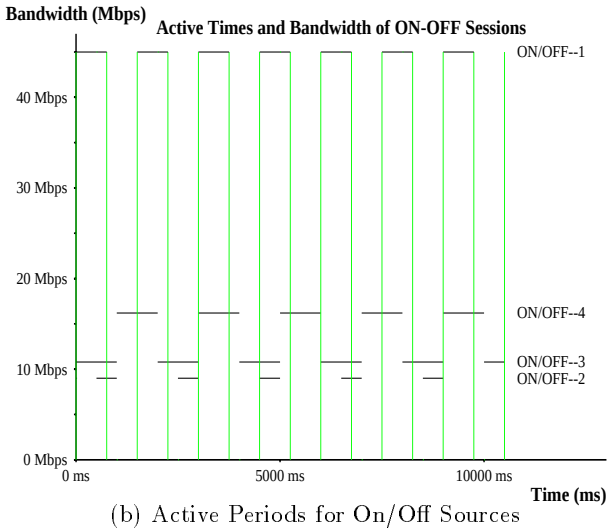
In this section we present simulation experiments to illustrate how H-WF^{2}Q+ can be used to support hierarchical link-sharing.

We consider the link-sharing structure shown in Fig. 8(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 alternative 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. 8(b).

Fig. 9 shows the bandwidth vs. time graphs for each of the TCP sessions under consideration. Fig. 9(a) shows the measured bandwidth curves with an H-WF^{2}Q+ server. The bandwidth is measured by exponentially averaging over 50ms windows. A close-up comparison between the measured bandwidth and the bandwidth under the ideal H-GPS server during the interval [4500 ms, 8500ms] is shown in Fig. 9(b). We notice that the measured bandwidth curves

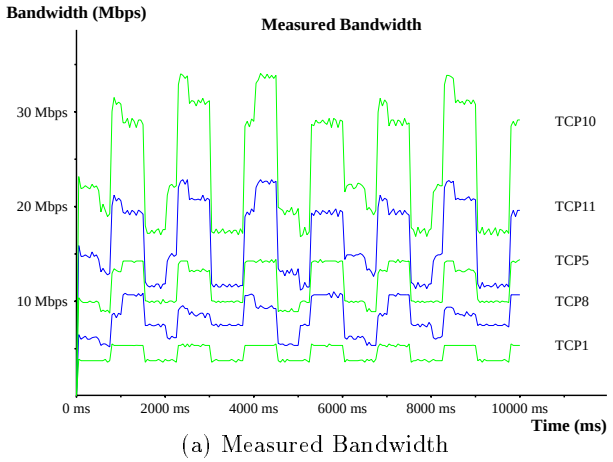


(a) Class Hierarchy

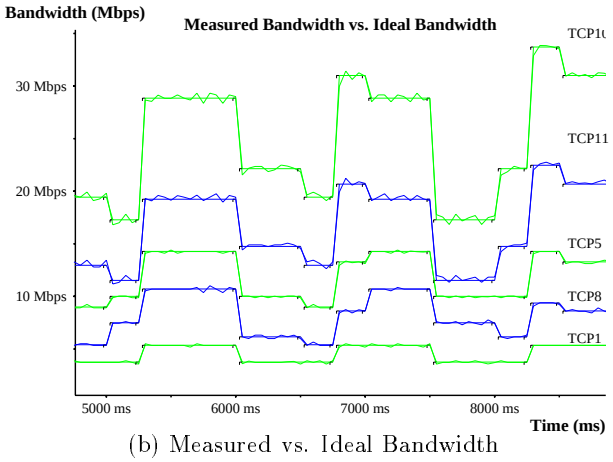


(b) Active Periods for On/Off Sources

Figure 8: Class Hierarchy and ON/OFF Sources



(a) Measured Bandwidth



(b) Measured vs. Ideal Bandwidth

Figure 9: Ideal and Measured link-Sharing Bandwidth under H-WF^{2}Q+

track very closely to the ideal bandwidth curves. Therefore, from the point of view of link-sharing, H-WF^{2}Q+ and H-GPS provide almost identical services.

Now we examine in more detail the dynamics of bandwidth sharing due to the alternation between active and idle states by on-off sources. At time 5000 ms, on-off source 4 becomes active while on-off sources 2 and 3 become idle. On the one hand, the bandwidths received by TCP session 10 and 11 decrease because they lose more bandwidth to on-off source 4 than they gain from on-off sources 2 and 3. On the other hand, the bandwidths of TCP sessions 5 and 8 increase as they benefit from the departure of on-off sources 2 and 3, but are not affected by on-off source 4.

At time 5250 ms, on-off source 1 becomes idle. Since it is at the first level in the hierarchy, its excess bandwidth is shared by all sessions, though with different relative ratio's. As can be seen, the bandwidths for all sessions increase due to the extra available bandwidth. When the on-off source 1 becomes active again at time 600 ms, the bandwidth of all sessions decrease to the original level before time 5250 ms. Similar oscillations happen at time pairs (6750ms,7500ms) and (8250ms,9000ms). Also note that on/off source 1 is the only session that affects the bandwidth of TCP session 1. This is because TCP session 1 is at the first level, and won't be affected by any changes in lower levels as long as N1 is backlogged.

At time 8000ms, on-off source 4 becomes idle while on-off source 3 becomes active. The bandwidth of TCP session 1 and 5 are not affected because there are no changes in their levels. The bandwidths of TCP session 10 and 11 increase since they now gain more from the departure of on-off source 4 than they lose to the arrival of on-off source 3. The bandwidth of TCP session 8 decreases due to the arrival of on-off source 3.

6 Related Work

In [16], the hierarchical WFQ mechanism 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 is performed only among non-real-time sessions but not between real-time and non-real-time sessions. That is, due to deficiencies of WFQ, tight delay bounds and accurate hierarchical link sharing

cannot be achieved simultaneously with H-WFQ.

In [13], an implementation of H-WFQ is presented. The scheduler implemented is not actually a 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 [8], Floyd and Jacobson present an architecture which during congestion first gives priority to link sharing, and once the link sharing goals are met, a general scheduler provides for the various service guarantees. Our work differs from this work in that we build our framework on H-PFQ, which has theoretically proven properties for supporting link-sharing, real-time service, and best-effort service. The Class Based Queueing (CBQ) algorithm used to realize the link-sharing architecture, however, is rather ad hoc. It is unclear whether the guaranteed real-time service can be provided within their framework.

A number of algorithms such as Self-Clocked Fair Queueing [5, 9], Stochastic Fair Queueing [12], Deficit Round Robin [17], and Frame-based Fair Queueing [18] 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.

7 Conclusion

We make three contributions in this paper. First, we develop techniques to analyze the delay and fairness properties of Hierarchical Packet Fair Queueing algorithms. 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. Second, we propose a new PFQ algorithm called WF^{2}Q+ 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). Finally, we present an implementation of H-WF^{2}Q+ that provides similar delay bounds and bandwidth distribution to those provided by the idealized H-GPS server. The proposed H-WF^{2}Q+ server 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.

In the paper we have focused on a specific type of H-PFQ server that has a tree structure. The work can easily be extended to the more generalized directed acyclic graph structure, where a node can receive its service from more than one parent node.

References

[1] J. Bennett and H. Zhang. Worst-case fair packet fair queueing algorithms. Technical report, 1996. Submitted for publication.

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

[3] 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.

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

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

[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] D. Ferrari and D. Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, 8(3):368–379, April 1990.

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

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

[10] P. Goyal, S. Lam, and H. Vin. Determining end-to-end delay bounds in heterogeneous networks. In *Proceedings of the 5th International Workshop on Network and Operating System Support For Digital Audio and Video*, pages 287–298, Durham, New Hampshire, April 1995.

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

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

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

[14] 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.

[15] 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.

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

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

[18] D. Stilliadis and A. Verma. Frame-based fair queueing: A new traffic scheduling algorithm for packet-switched networks. Technical Report USCS-CRL-95-39, University of California at Santa Cruz, July 1995.

[19] J. Turner. New directions in communications(or which way to the information age?). *IEEE Communication Magazine*, 24(10), October 1986.

[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 $[t_1, t_2]$ be any time period that session i is continuously backlogged. It immediately follows that the logical queue at node $p^h(i)$ is continuously backlogged 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

$$W_{p^h(i)}(t_1,t_2) \geq \frac{\phi_{p^h(i)}}{\phi_{p^{h+1}(i)}} W_{p^{h+1}(i)}(t_1,t_2) - \alpha_{p^h(i)} \qquad (33)$$

where $W_{p^h(i)}(t_1,t_2)$ is the amount of service received by node $p^h(i)$ in $[t_1,t_2]$.

By multiplying $\frac{\phi_i}{\phi_{p^h(i)}}$ at both sides of (33), we have

$$\frac{\phi_i}{\phi_{p^h(i)}} W_{p^h(i)}(t_1,t_2) \geq \frac{\phi_i}{\phi_{p^{h+1}(i)}} W_{p^{h+1}(i)}(t_1,t_2) - \frac{\phi_i}{\phi_{p^h(i)}} \alpha_{p^h(i)} \qquad (34)$$

Adding (34) for h=0, $\cdots$, H-1, and eliminating common terms on both sides, we have:

$$\begin{aligned} W_i(t_1,t_2) &\geq \frac{\phi_i}{\phi_{p^H(i)}} W_{p^H(i)}(t_1,t_2) - \sum_{h=0}^{H-1} \frac{\phi_i}{\phi_{p^h(i)}} \alpha_{p^h(i)} \\ &= \phi_i r(t_2-t_1) - \sum_{h=0}^{H-1} \frac{\phi_i}{\phi_{p^h(i)}} \alpha_{p^h(i)} \\ &= r_i(t_2-t_1) - \sum_{h=0}^{H-1} \frac{\phi_i}{\phi_{p^h(i)}} \alpha_{p^h(i)} \qquad (35) \end{aligned}$$

Therefore, the NB-WFI for session i in the H-PFQ server is

$$\alpha_{i,H-PFQ} = \sum_{h=0}^{H-1} \frac{\phi_i}{\phi_{p^h(i)}} \alpha_{p^h(i)} \qquad (36)$$

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

$$W_i(t_1,d_i^k) \geq \frac{\phi_i}{\phi_p(i)} W_{p(i)}(t_1,d_i^k) - (r_i D_i - \sigma_i) \qquad (37)$$

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 the following holds

$$W_{p^h(i)}(t_1,d_i^k) \geq \frac{\phi_{p^h(i)}}{\phi_{p^{h+1}(i)}} W_{p^{h+1}(i)}(t_1,t_2) - \alpha_{p^h(i)} \qquad (38)$$

Multiplying $\frac{\phi_{p(i)}}{\phi_{p^h(i)}}$ at both sides of (38), we have

$$\frac{\phi_{p(i}}{\phi_{p^h(i)}} W_{p^h(i)}(t_1,d_i^k) \geq \frac{\phi_{p(i)}}{\phi_{p^{h+1}(i)}} W_{p^{h+1}(i)}(t_1,d_i^k) - \frac{\phi_{p(i)}}{\phi_{p^h(i)}} \alpha_{p^h(i)} \qquad (39)$$

Summing (39) for h=1, $\cdots$, H-1, and eliminating common terms on both sides, we have:

$$\begin{aligned} W_{p(i)}(t_1,d_i^k) &\geq \frac{\phi_{p(i)}}{\phi_{p^H(i)}} W_{p^H(i)}(t_1,d_i^k) - \sum_{h=1}^{H-1} \frac{\phi_{p(i)}}{\phi_{p^h(i)}} \alpha_{p^h(i)} \\ &= \phi_{p(i)} r(d_i^k - t_1) - \sum_{h=1}^{H-1} \frac{\phi_{p(i)}}{\phi_{p^h(i)}} \alpha_{p^h(i)} \\ &= r_{p(i)}(d_i^k - t_1) - \sum_{h=1}^{H-1} \frac{\phi_{p(i)}}{\phi_{p^h(i)}} \alpha_{p^h(i)} \qquad (40) \end{aligned}$$

Plugging into (37), we have

$$W_i(t_1,d_i^k) \geq r_i(d_i^k - t_1) - \sum_{h=1}^{H-1} \frac{\phi_i}{\phi_{p^h(i)}} \alpha_{p^h(i)} - r_i D_i + \sigma_i \qquad (41)$$

Since session i queue is FIFO and leaky bucket constrained, (19) and (20) holds. Combining them with (41), we have

$$\sigma_i + r_i(a_i^k - t_1) \geq r_i(d_i^k - t_1) - \sum_{h=1}^{H-1} \frac{\phi_i}{\phi_{p^h(i)}} \alpha_{p^h(i)} - r_i D_i + \sigma_i \qquad (42)$$

Rearranging terms and dividing both sides by r_i, we have

$$d_i^k - a_i^k \leq D_i + \sum_{h=1}^{H-1} \frac{\alpha_{p^h(i)}}{r_{p^h(i)}} \qquad (43)$$

Q.E.D.