Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks*

Pawan Goyal, Harrick M. Vin, and **Haichen Cheng**

Distributed Multimedia Computing Laboratory
Department of Computer Sciences, University of Texas at Austin
Taylor Hall 2.124, Austin, Texas 78712-1188
E-mail: {pawang,vin,hccheng}@cs.utexas.edu, Telephone: (512) 471-9732, Fax: (512) 471-8885
URL: http://www.cs.utexas.edu/users/dmcl

Abstract

We present Start-time Fair Queuing (SFQ) algorithm that is computationally efficient, achieves fairness regardless of variation in a server capacity, and has the smallest fairness measure among all known fair scheduling algorithms. We analyze its throughput, single server delay, and end-to-end delay guarantee for variable rate Fluctuation Constrained (FC) and Exponentially Bounded Fluctuation (EBF) servers. We show that SFQ is better suited than Weighted Fair Queuing for integrated services networks and it is strictly better than Self Clocked Fair Queuing. To support heterogeneous services and multiple protocol families in integrated services networks, we present a hierarchical SFQ scheduler and derive its performance bounds. Our analysis demonstrates that SFQ is suitable for integrated services networks since it: (1) achieves low average as well as maximum delay for low-throughput applications (e.g., interactive audio, telnet, etc.); (2) provides fairness which is desirable for VBR video; (3) provides fairness, regardless of variation in a server capacity, for throughput-intensive, flow-controlled data applications; (4) enables hierarchical link sharing which is desirable for managing heterogeneity; and (5) is computationally efficient.

1 Introduction

1.1 Motivation

Integrated services networks are required to support a variety of applications (e.g., audio and video conferencing, multimedia information retrieval, ftp, telnet, WWW, etc.) with a wide range of Quality of Service (QoS) requirements. Whereas continuous media applications such as audio and video conferencing require the network to provide QoS guarantees with respect to bandwidth, packet delay, and loss; applications such as telnet and WWW require low packet delay and loss. Throughput intensive applications like ftp, on the other hand, require network resources to be allocated such that the throughput is maximized. A network meets these requirements primarily by appropriately *scheduling* its resources.

* This research was supported in part by IBM Graduate Fellowship, IBM Faculty Development Award, Intel, the National Science Foundation (Research Initiation Award CCR-9409666), NASA, Mitsubishi Electric Research Laboratories (MERL), and Sun Microsystems Inc.

To appear in the Proceedings of SIGCOMM 1996.

To determine the characteristics of a suitable scheduling algorithm, consider the requirements of some of the principal applications envisioned for integrated services networks:

- *Audio applications*: To maintain adequate interactivity for such applications, scheduling algorithms must provide low average and maximum delay.
- *Video applications*: Variable bit rate (VBR) video sources, which are expected to impose significant requirements on network resources, have unpredictable as well as highly variable bit rate requirement at multiple time-scales [10, 12, 14]. These features impose two key requirements on network resource management:
  - Due to the difficulty in predicting the bit rate requirement of VBR video sources, video channels may utilize more than the reserved bandwidth. As long as the additional bandwidth used is not at the expense of other channels (i.e., if the channel utilizes idle bandwidth), it should not be penalized in the future by reducing its bandwidth allocation.
  - Due to multiple time-scale variation in the bit rate requirement of video sources, to achieve efficient utilization of resources, a network will have to over-book available bandwidth. Since such over-booking may yield persistent congestion, a network should provide some QoS guarantees even in the presence of congestion.

  Unfair scheduling algorithms, such as Virtual Clock [24], Delay EDD [23], etc., penalize channels for the use of idle bandwidth and do not provide any QoS guarantee in the presence of congestion [18]. Fair scheduling algorithms, on the other hand, guarantee that, regardless of prior usage or congestion, bandwidth would be allocated fairly[18]. Hence, fair scheduling algorithms are desirable for video applications.
- *Data applications*: To support low-throughput, interactive data applications (e.g., telnet), scheduling algorithms must provide low average delay. On the other hand, to support throughput-intensive, flow-controlled applications in heterogeneous, large-scale, decentralized networks, scheduling algorithms must allocate bandwidth fairly [5, 15, 19]. Due to the coexistence of VBR video sources and data sources in integrated services networks, the bandwidth available to data applications may vary significantly over time. Consequently, the fairness property of the scheduling algorithm must hold regardless of variation in server capacity.

Hence, in summary, a suitable scheduling algorithm for integrated services networks should: (1) achieve low average as well as maximum delay for low throughput applications (e.g., interactive audio, telnet, etc.); (2) provide fairness for VBR video; and (3) provide fairness, regardless of variation in server capacity, for throughput-intensive, flow-controlled data applications. Furthermore, since such networks will support a wide variety of services and multiple protocol families, the scheduling algorithm should facilitate hierarchical link sharing [6, 20]. Finally, to facilitate its implementation in high-speed networks, it should be computationally efficient. A scheduling algorithm that achieves all of these objectives is the subject of investigation in this paper.

1.2 Relation To Previous Work

Each unit of data transmission at the network level is a packet. We refer to the sequence of packets transmitted by a source as a *flow* [24]. Each packet within a flow is serviced by a sequence of servers (or switching elements) along the path from the source to the destination in the network. Before we describe fair scheduling algorithms that may be employed by the servers, let us consider the precise meaning of fair allocation of link bandwidth.

Intuitively, allocation of link bandwidth is fair if equal bandwidth is allocated in every time interval to all the flows. This concept generalizes to weighted fairness in which the bandwidth must be allocated in proportion to the *weights* associated with the flows. Formally, if r_f is the weight of flow f and $W_f(t_1,t_2)$ is the aggregate service (in bits) received by it in the interval $[t_1,t_2]$, then an allocation is fair if, for all intervals $[t_1,t_2]$ in which both flows f and m are backlogged, $\frac{W_f(t_1,t_2)}{r_f} - \frac{W_m(t_1,t_2)}{r_m} = 0$. Clearly, this is an idealized definition of fairness as it assumes that flows can be served in infinitesimally divisible units. The objective of fair *packet* scheduling algorithms is to ensure that $\mid \frac{W_f(t_1,t_2)}{r_f} - \frac{W_m(t_1,t_2)}{r_m} \mid$ is as close to 0 as possible. However, it has been shown in [7] that if a packet scheduling algorithm guarantees that $\mid \frac{W_f(t_1,t_2)}{r_f} - \frac{W_m(t_1,t_2)}{r_m} \mid \leq H(f,m)$ for all intervals $[t_1,t_2]$ then $H(f,m) \geq \frac{1}{2}\left(\frac{l_f^{max}}{r_f} + \frac{l_m^{max}}{r_m}\right)$, where $H(f,m)$ is a function of the properties of flows f and m and l_f^{max} and l_m^{max} denote the maximum lengths of packets of flow f and m, respectively.

Several fair scheduling algorithms that achieve value of $H(f,m)$ close to the lower bound have been proposed in the literature. The earliest known fair scheduling algorithm is Weighted Fair Queuing (WFQ) [5] (also referred to as Packet-by-Packet Generalized Processor Sharing (PGPS) [18]). WFQ was designed to emulate a hypothetical bit-by-bit weighted round robin server in which the number of bits of a flow served in a round is proportional to the weight of the flow. Since packets can not be serviced a bit at a time, WFQ emulates bit-by-bit round robin by scheduling packets in the increasing order of their departure times in the hypothetical server. To compute this departure order, WFQ associates two tags, a *start tag* and a *finish tag*, with every packet of a flow. Specifically, if p_f^j and l_f^j denote the j^{th} packet of flow f and its length, respectively, and if $A(p_f^j)$ denotes the arrival time of packet p_f^j at the server, then start tag $S(p_f^j)$ and finish tag $F(p_f^j)$ of packet p_f^j are defined as:

$$S(p_f^j) = \max\{v(A(p_f^j)), F(p_f^{j-1})\} \quad j \geq 1 \tag{1}$$

$$F(p_f^j) = S(p_f^j) + \frac{l_f^j}{r_f} \quad j \geq 1 \tag{2}$$

where $F(p_f^0) = 0$ and $v(t)$ is defined as the round number that would be in progress at time t if the packets were being serviced in a bit-by-bit weighted round robin manner. Formally, $v(t)$ is defined as:

$$\frac{dv(t)}{dt} = \frac{C}{\sum_{j \in B(t)} r_j} \tag{3}$$

where C is the capacity of the server and $B(t)$ is the set of backlogged flows at time t in the bit-by-bit round robin server. WFQ then schedules packets in the increasing order of their finish tags.

Observe that implementation of WFQ requires computation of $v(t)$, which in turn requires simulation of bit-by-bit round robin server in real time. This simulation is computationally expensive. Furthermore, as the following examples illustrate, since WFQ is based on the assumption that the capacity of a server is constant, it fails to provide fairness over servers with time varying capacity.

Example 1 *Let the capacity of the server that WFQ is emulating be C pkts/s, $C > 1$. Let the actual server capacity be 1 pkt/s in [0,1) and C pkt/s in [1,2). Consider two flows f and m both of which have unit length packets and weights of 1 pkt/s. Let flow f send $C+1$ packets at time 0. Hence, for flow f, $F(p_f^j) = j$; $1 \leq j \leq C+1$. Let flow m become backlogged at $t = 1$ and be backlogged during the interval [1,2]. Since only flow f is backloged during [0,1), using (3), we get $v(1) = C$. Hence, for flow m, $F(p_m^1) = C+1$. Since WFQ schedules packets in the increasing order of finish tags, we get: $C-1 \leq W_f(1,2) \leq C$ and $W_m(1,2) \leq 1$. However, for fair allocation of bandwidth, $W_f(1,2)$ and $W_m(1,2)$ should both be $C/2$. Since C can be chosen arbitrarily, this example illustrates the unfairness that can result when the actual capacity is lower than the capacity being assumed.*

Example 2 *Let the capacity of the server that WFQ is emulating be 1 pkts/s. Let the actual server capacity be C pkt/s, C > 1, in [0,1) and 1 pkt/s in [1,C). Consider two flows f and m both of which have weights of 1 pkt/s. Let flow f send $\frac{3C}{2}$ packets at time 0. Hence, for flow f, $F(p_f^j) = j$; $1 \leq j \leq \frac{3C}{2}$. Let flow m send C packets at time 1. Since only flow f is backloged during [0,1), $v(1) = 1$. Hence, for flow m, $F(p_m^j) = j+1$; $1 \leq j \leq C$. Since WFQ schedules packets in the increasing order of finish tags, while $W_f(1, C+1) \leq 1$, $C-1 \leq W_m(1, C+1) \leq C$. However, for fair allocation of bandwidth, $W_f(1,C+1)$ and $W_m(1,C+1)$ both should be $C/2$. Since, C can be chosen arbitrarily, this example illustrates the unfairness that can result when the actual capacity is greater than the capacity being assumed.*

Thus, WFQ fails to provide fairness over variable rate servers. As we will outline in Section 3, to be useful for hierarchical link sharing [6, 20], a scheduling algorithm must provide fairness over variable rate servers. Consequently, WFQ fails to meet two key requirements (i.e., fairness over variable rate servers and support for hierarchical link sharing) of a fair scheduling algorithm for integrated services networks. Observe that it may be possible to extend WFQ to provide fairness over variable rate servers by changing the definition of $v(t)$ as given in (3) to be a function of time varying server capacity $C(t)$. However, due to the unpredictable and multiple time-scale variation in VBR video bit rate, it may not be possible to accurately estimate $C(t)$. Furthermore, this would make the computation of $v(t)$ even more expensive, thereby making WFQ infeasible for high speed networks.

Fair Queuing based on Start-time (FQS), proposed in [13], computes start tag and finish tag of a packet exactly as in WFQ. However, instead of scheduling packets in the increasing order of finish tags, it schedules packets in the increasing order of start tags. Though FQS has advantages for processor scheduling, it is not known to have any advantage over WFQ for scheduling packets in a network. Moreover, since it utilizes $v(t)$ as defined in (3), it has all the disadvantages of WFQ.

Self Clocked Fair Queuing (SCFQ), originally proposed in [4] and later analyzed in [7], was designed to reduce the computational complexity of fair scheduling algorithms like WFQ. SCFQ also schedules packets in the increasing order of finish tags. However, it achieves efficiency over WFQ by approximating $v(t)$ with the finish tag of the packet in service at time t. It has been shown that the value of $H(f,m)$ for SCFQ is $(\frac{l_f^{max}}{r_f} + \frac{l_m^{max}}{r_m})$, which is only a factor of two away from the lower bound [7]. The main limitation of SCFQ is that it increases the maximum delay incurred by the packets significantly. Specifically, if Q is the set of flows served by a server and C its capacity, then packets of flow f may incur $\frac{\sum_{n\in Q \wedge n\neq f} l_n^{max}}{C}$ more delay in SCFQ than in WFQ [8]. This may be unacceptably large in many cases.

WFQ and SCFQ sort and schedule packets in the increasing order of finish tags. Hence, per packet computational complexity is $O(logQ)$ where Q is the number of flows served by the server. To reduce this per packet computational complexity, Deficit Round Robin (DRR) was proposed in [21]. It is a derivative of weighted round robin algorithm designed to accommodate variable length packets of a flow. Though the per packet computational complexity of DRR is $O(1)$ per packet, it has the following two limitations:

1. The value of $H(f,m)$ for DRR is $\left(1 + \frac{l_f^{max}}{r_f} + \frac{l_m^{max}}{r_m}\right)$ when $\min_{n\in Q} r_n = 1$, which can deviate arbitrarily away from that of fair scheduling algorithms like SCFQ. For instance, if $r_f = r_m = 100$ and $l_f^{max} = l_m^{max} = 1$, then $H(f,m)$ for DRR is 1.02, which is 50 times larger than the corresponding 0.02 value for SCFQ. Clearly, appropriate choice of weights can make this factor as high as desired. Since absolute values of $H(f,m)$ have no physical meaning, the relative values are important.

2. The maximum delay incurred by packets serviced by a DRR server can be arbitrarily high. To observe this, consider a weighted round robin server with all packets of size l (DRR is equivalent to weighted round robin server in such a scenario). It is easily observed that a lower bound on maximum delay of a packet of flow f is $\frac{\sum_{n\in Q \wedge n\neq f} l * r_n}{C}$. Since weights can be arbitrary, the lower bound can be arbitrarily high.

In summary, the design of a fair scheduling algorithm that is: (1) computationally efficient, (2) provides fairness over variable rate servers, (3) facilitates hierarchical link sharing, and (4) has good delay properties is an open problem.

1.3 Research Contributions of This Paper

In this paper, we present Start-time Fair Queuing (SFQ) algorithm that is computationally efficient and allocates bandwidth fairly regardless of variation in a server rate. We show that it has a fairness measure of $(\frac{l_f^{max}}{r_f} + \frac{l_m^{max}}{r_m})$, which is only a factor two away from the lower bound and is at least as good as the fairness measure of all known fair scheduling algorithms. We analyze the throughput, single server delay, and end-to-end delay guarantee of SFQ. To accommodate links whose capacity fluctuates over time (for example, flow-controlled and broadcast medium links), this analysis is carried out for servers which can be modeled as either Fluctuation Constrained (FC) or Exponentially Bounded Fluctuation (EBF) servers [17]. To the best of our knowledge, this is the first analysis of a fair or a real-time scheduling algorithm for such servers. To support hierarchical link sharing, we present a hierarchical SFQ scheduler. We build upon the analysis of FC and EBF servers and analyze the throughput, delay and end-to-end delay guarantees of a flow when the link bandwidth is hierarchically partitioned. The analysis is simple and conceptually elegant. We demonstrate that the hierarchical SFQ scheduler, in addition to supporting heterogeneity, can be used to achieve separation of delay and throughput allocation as well as delay shifting (i.e., reduction of delay of a set of flows while increasing the delay of other flows).

Our analysis demonstrates that: (1) SFQ is better suited than WFQ for integrated services networks since it efficiently achieves fairness over variable rate servers and provides significantly smaller average and maximum packet delay to low-throughput applications; (2) SFQ is strictly better than SCFQ since maximum packet delay in SFQ is considerably smaller than in SCFQ and both have same the fairness measure and implementation complexity; (3) SFQ is strictly better than FQS since it has lower complexity and achieves fairness over variable rate servers without increasing the maximum packet delay; and (4) SFQ provides considerably better fairness properties and smaller maximum delay than DRR.

We demonstrate that SFQ is suitable for integrated services networks since it: (1) achieves low average as well as maximum delay for low-throughput applications (e.g., interactive audio, telnet, etc.); (2) provides fairness which is desirable for VBR video; (3) provides fairness, regardless of variation in server capacity, for throughput-intensive, flow-controlled data applications; (4) enables hierarchical link sharing which is desirable for managing heterogeneity; and (5) is computationally efficient.

The rest of the paper is structured as follows. We present SFQ algorithm and analyze its fairness, throughput, single server delay guarantee, and end-to-end delay guarantee in Section 2. We discuss hierarchical link sharing in Section 3 and present our implementation of SFQ for an ATM network interface in Solaris 2.4 environment in Section 4. Finally, Section 5 summarizes our results.

2 Start-time Fair Queuing

In Start-time Fair Queuing (SFQ) algorithm, two tags, a start tag and a finish tag, are associated with each packet. However, unlike WFQ and SCFQ, packets are scheduled in the increasing order of the start tags of the packets. Furthermore, $v(t)$ is defined as the start tag of the packet in service at time t. The complete algorithm is defined as follows:

1. On arrival, a packet p_f^j is stamped with start tag $S(p_f^j)$, computed as:

$$S(p_f^j) = \max\{v(A(p_f^j)), F(p_f^{j-1})\} \quad j \geq 1 \tag{4}$$

where $F(p_f^j)$, the finish tag of packet p_f^j, is defined as:

$$F(p_f^j) = S(p_f^j) + \frac{l_f^j}{r_f} \quad j \geq 1 \tag{5}$$

where $F(p_f^0) = 0$ and r_f is the weight of flow f.

2. Initially the server virtual time is 0. During a busy period, the server virtual time at time t, $v(t)$, is defined to be equal to the start tag of the packet in service at time t[1]. At the end of a busy period, $v(t)$ is set to the maximum of finish tag assigned to any packets that have been serviced by time t[2].

3. Packets are serviced in increasing order of the start tags; ties are broken arbitrarily (some tie breaking rules may be more desirable than others and are discussed in Section 2.3).

[1] Observe that server virtual time changes only when a packet finishes service.

[2] We set $v(t)$ to the maximum of finish tags of the packets at the end of busy period only for clarity of proofs; all the start tags as well as the server virtual time can be equivalently set to zero.

As is evident from the definition, the computation of $v(t)$ in SFQ is inexpensive since it only involves examining the start tag of packet in service. Hence, the computational complexity of SFQ is same as SCFQ, which is $O(logQ)$ per packet, where Q is the number of flows at the server.

Traditionally, scheduling algorithms have been analyzed only for servers whose service rate does not vary over time. However, service rate of flow-controlled, broadcast medium and wireless links may fluctuate over time. Fluctuation in service rate may also occur due to variability in CPU capacity available for processing packets (for example, a CPU constrained IP router may not have sufficient CPU capacity to process packets when routing updates occur). If a server is shared by multiple types of traffic with some traffic types being given priority over the other, then for lower priority traffic, the link appears as a server with fluctuating service rate. In order to accommodate such scenarios, we analyze SFQ for servers with bounded fluctuation in service rate.

Two server models, termed *Fluctuation Constrained (FC)* server and *Exponentially Bounded Fluctuation (EBF)* server, that have bounded fluctuation in service rate and are suitable for modeling many variable rate servers have been introduced in [17]. A FC server has two parameters; average rate C (bits/s) and burstiness $\delta(C)$ (bits). Intuitively, an FC server, in any interval during a busy period, does at most $\delta(C)$ less work than an equivalent constant rate server. Formally,

Definition 1 *A server is a Fluctuation Constrained (FC) server with parameters $(C, \delta(C))$, if for all intervals $[t_1, t_2]$ in a busy period of the server, the work done by the server, denoted by $W(t_1, t_2)$, satisfies:*

$$W(t_1, t_2) \geq C(t_2 - t_1) - \delta(C) \tag{6}$$

EBF server is a stochastic relaxation of FC server. Intuitively, the probability of work done by an EBF server deviating from the average rate by more than γ, decreases exponentially with γ [3]. Formally,

Definition 2 *A server is an Exponentially Bounded Fluctuation (EBF) server with parameters $(C, B, \alpha, \delta(C))$, if for all intervals $[t_1, t_2]$ in a busy period of the server, the work done by the server, denoted by $W(t_1, t_2)$, satisfies:*

$$P(W(t_1, t_2) < C(t_2 - t_1) - \delta(C) - \gamma) \leq Be^{-\alpha\gamma} \quad 0 \leq \gamma \tag{7}$$

In what follows, we analyze the fairness of SFQ for any variable rate server, and its throughput and delay guarantees for FC and EBF servers. Since a $(C, 0)$ FC server is a constant rate server, the following analysis is also valid for constant rate servers. We present the proofs of only few results; the rest of the proofs are presented in [11].

2.1 Fairness Guarantee

To prove that SFQ is fair, we need to prove a bound on $\left| \frac{W_f(t_1,t_2)}{r_f} - \frac{W_m(t_1,t_2)}{r_m} \right|$ for any interval in which both flows f and m are backlogged. We achieve this objective by establishing a lower and an upper bound on $W_f(t_1, t_2)$ in Lemmas 1 and 2, respectively.

Lemma 1 *If flow f is backlogged throughout the interval $[t_1, t_2]$, then in a SFQ server:*

$$r_f(v_2 - v_1) - l_f^{max} \leq W_f(t_1, t_2) \tag{8}$$

where $v_1 = v(t_1)$ and $v_2 = v(t_2)$.

[3] The EBF server as presented here has an extra parameter $\delta(C)$. However, this parameter does not change the definition significantly.

Proof: Since $W_f(t_1, t_2) \geq 0$, if $r_f(v_2 - v_1) - l_f^{max} \leq 0$, (8) holds trivially. Hence, consider the case where $r_f(v_2 - v_1) - l_f^{max} > 0$, i.e., $v_2 > v_1 + \frac{l_f^{max}}{r_f}$. Let packet p_f^k be the first packet of flow f, that receives service in the open interval (v_1, v_2). To observe that such a packet exists, consider the following two cases:

- Packet p_f^n such that $S(p_f^n) < v_1$ and $F(p_f^n) > v_1$ exists: Since flow f is backlogged in $[t_1, t_2]$, we conclude $v(A(p_f^{n+1})) \leq v_1$. From (4) and (5), we get:

$$S(p_f^{n+1}) = F(p_f^n) \tag{9}$$

  Since $F(p_f^n) \leq S(p_f^n) + \frac{l_f^{max}}{r_f}$ and $S(p_f^n) < v_1$, we get:

$$\begin{aligned} S(p_f^{n+1}) &< v_1 + \frac{l_f^{max}}{r_f} &(10)\\ &< v_2 &(11) \end{aligned}$$

  Since $S(p_f^{n+1}) = F(p_f^n) > v_1$, using (11), we conclude $S(p_f^{n+1}) \in (v_1, v_2)$.

- Packet p_f^n such that $S(p_f^n) = v_1$ exists: p_f^n may finish service at time $t < t_1$ or $t \geq t_1$. In either case, since flow f is backlogged in $[t_1, t_2]$, $v(A(p_f^{n+1})) \leq v_1$. Hence, $S(p_f^{n+1}) = F(p_f^n)$. Since $F(p_f^n) \leq S(p_f^n) + \frac{l_f^{max}}{r_f}$ and $S(p_f^n) = v_1$, we get:

$$\begin{aligned} S(p_f^{n+1}) &\leq v_1 + \frac{l_f^{max}}{r_f} &(12)\\ &< v_2 &(13) \end{aligned}$$

  Since $S(p_f^{n+1}) = F(p_f^n) > v_1$, using (13), we conclude $S(p_f^{n+1}) \in (v_1, v_2)$.

Since either of the two cases always holds, we conclude that packet p_f^k such that $S(p_f^k) \in (v_1, v_2)$ exists. Furthermore, from (10) and (12), we get:

$$S(p_f^k) \leq v_1 + \frac{l_f^{max}}{r_f} \tag{14}$$

Let p_f^{k+m} be the last packet to receive service in the virtual time interval (v_1, v_2). Hence,

$$F(p_f^{k+m}) \geq v_2 \tag{15}$$

From (14) and (15), we conclude:

$$F(p_f^{k+m}) - S(p_f^k) \geq (v_2 - v_1) - \frac{l_f^{max}}{r_f} \tag{16}$$

But since flow f is backlogged in the interval (v_1, v_2), from (4) and (5) we know:

$$\begin{aligned} F(p_f^{k+m}) &= S(p_f^k) + \sum_{n=0}^{n=m} \frac{l_f^{k+n}}{r_f} &(17)\\ F(p_f^{k+m}) - S(p_f^k) &= \sum_{n=0}^{n=m} \frac{l_f^{k+n}}{r_f} &(18) \end{aligned}$$

Hence, from (18) and (16), we get:

$$\begin{aligned} \sum_{n=0}^{n=m} \frac{l_f^{k+n}}{r_f} &\geq (v_2 - v_1) - \frac{l_f^{max}}{r_f} &(19)\\ \sum_{n=0}^{n=m} l_f^{k+n} &\geq r_f(v_2 - v_1) - l_f^{max} &(20) \end{aligned}$$

Since $S(p_f^{k+m}) < v_2$, packet p_f^{k+m} is guaranteed to have been transmitted by t_2. Hence, $W_f(t_1, t_2) \geq \sum_{n=0}^{n=m} l_f^{k+n}$, and the lemma follows. ■

Lemma 2 *In a SFQ server, during any interval* $[t_1, t_2]$*:*

$$W_f(t_1, t_2) \leq r_f(v_2 - v_1) + l_f^{max} \tag{21}$$

where $v_1 = v(t_1)$ *and* $v_2 = v(t_2)$*.*

Proof: From the definition of SFQ, the set of flow f packets served in the interval $[v_1, v_2]$ have service tag at least v_1 and at most v_2. Hence, the set can be partitioned into two sets:

- Set D consisting of packets that have service tag at least v_1 and finish time at most v_2. Formally,

  $$D = \{k | v_1 \leq S(p_f^k) \leq v_2 \wedge F(p_f^k) \leq v_2\} \tag{22}$$

  From (4) and (5), we conclude:

  $$\sum_{k \in D} l_f^k \leq r_f(v_2 - v_1) \tag{23}$$

- Set E consisting of packets that have service tag at most v_2 and finish time greater than v_2. Formally,

  $$E = \{k | v_1 \leq S(p_f^k) \leq v_2 \wedge F(p_f^k) > v_2\} \tag{24}$$

  Clearly, at most one packet can belong to this set. Hence,

  $$\sum_{k \in E} l_f^k \leq l_f^{max} \tag{25}$$

From (23) and (25) we conclude that (21) holds. ■

Since unfairness between two flows in any interval is maximum when one flow receives maximum possible service and the other minimum service, Theorem 1 follows directly from Lemmas 1 and 2.

Theorem 1 *For any interval* $[t_1, t_2]$ *in which flows* f *and* m *are backlogged during the entire interval, the difference in the service received by two flows at a SFQ server is given as:*

$$\left| \frac{W_f(t_1, t_2)}{r_f} - \frac{W_m(t_1, t_2)}{r_m} \right| \leq \frac{l_f^{max}}{r_f} + \frac{l_m^{max}}{r_m} \tag{26}$$

Theorem 1 demonstrates that fairness measure of SFQ is at most a factor of 2 away from an optimal fair packet scheduling algorithm. Furthermore, it demonstrates that SFQ has the smallest fairness measure among all the known scheduling algorithms [22].

Observe that to establish Theorem 1, we did not make any assumptions about the service rate of the server. Hence, Theorem 1 holds regardless of the characteristics of the server. This demonstrates that SFQ achieves fair allocation of bandwidth over variable rate servers, and thus meets a fundamental requirement of fair scheduling algorithms for integrated services networks. This is an important advantage over WFQ that does not allocate bandwidth fairly over variable rate servers.

To experimentally evaluate the relative performance of SFQ and WFQ, we simulated 3 flows with the same destination traversing a single switch using REAL network simulator (see Figure 1). Source 1 transmitted a MPEG compressed VBR video sequence with average rate 1.21 Mb/s using 50 bytes packets [4]. Sources 2 and 3

[4] The video sequence was derived by digitizing and compressing television serial Frasier.

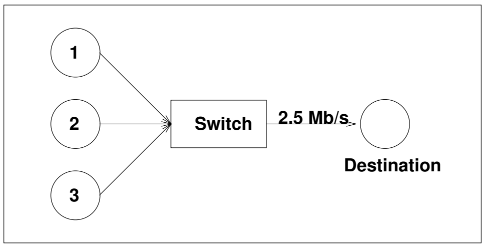

Figure 1 : The network topology used for the simulations

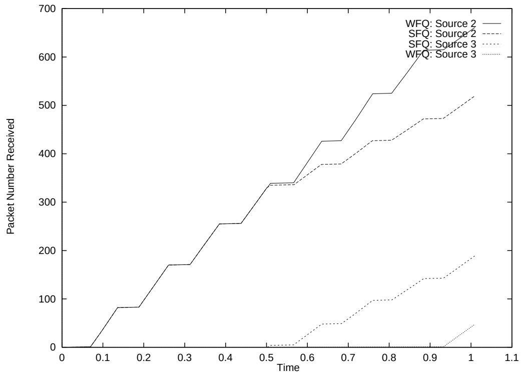

Figure 2 : Comparison of the number of packets received by sources 2 and 3 in WFQ and SFQ

were TCP Reno sources and used 200 bytes packets. The link capacity between the switch and the destination was 2.5 Mb/s. The scheduling algorithm at the switch gave higher priority to source 1 packets and scheduled source 2 and 3 packets using either WFQ or SFQ. Consequently, the output link at the switch appeared as a variable rate server to sources 2 and 3. The WFQ implementation used the link capacity to compute the finish tags. Source 3 was made active 500 ms after sources 1 and 2, and the network was simulated for one second. Figure 2 plots the sequence number of packets of sources 2 and 3 received by the destination when WFQ and SFQ were used. As the figure demonstrates, source 2 received unfair advantage over source 3 when WFQ was used. Specifically, when WFQ was used, during the 500ms interval after source 3 became active, whereas the destination received only 48 packets from source 3, it received 331 packets from source 2. On the other hand, when SFQ was used, it received 189 packets and 190 packets from sources 2 and 3, respectively. Furthermore, during the first 435 ms after source 3 was started, the number of source 3 packets received by the destination in WFQ and SFQ was 2 and 145, respectively. This experimentally validates the superiority of SFQ over WFQ for achieving fair allocation of bandwidth over variable rate servers.

2.2 Throughput Guarantee

In the previous sections, we have not assigned any interpretation to the weight of a flow. To establish the throughput and delay guarantee of a flow, we will henceforth interpret r_f as the rate assigned to flow f. Theorems 2 and 3 establish the throughput guaranteed to a flow by a SFQ FC and EBF server, respectively, when appropriate admission control procedures are used.

Theorem 2 *If* Q *is the set of flows served by a SFQ FC server with parameters* $(C, \delta(C))$*, and* $\sum_{n \in Q} r_n \leq C$*, then for all intervals* $[t_1, t_2]$ *in which flow* f *is backlogged throughout the interval,*

$W_f(t_1, t_2)$ *is given as:*

$$W_f(t_1,t_2) \geq r_f(t_2 - t_1) - r_f\frac{\sum_{n\in Q} l_n^{max}}{C} - r_f\frac{\delta(C)}{C} - l_f^{max} \tag{27}$$

Proof: Let $v(t_1) = v_1$ and let $\widehat{W}(v_1, v_2)$ denote the aggregate length of packets served by the server in the virtual time interval $[v_1, v_2]$. Then, from Lemma 2 we conclude:

$$\widehat{W}(v_1,v_2) \leq \sum_{n\in Q} r_n(v_2 - v_1) + \sum_{n\in Q} l_n^{max} \tag{28}$$

Since $\sum_{n\in Q} r_n \leq C$,

$$\widehat{W}(v_1,v_2) \leq C(v_2 - v_1) + \sum_{n\in Q} l_n^{max} \tag{29}$$

Define v_2 as:

$$v_2 = v_1 + t_2 - t_1 - \frac{\sum_{n\in Q} l_n^{max}}{C} - \frac{\delta(C)}{C} \tag{30}$$

Then from (29), we conclude:

$$\begin{aligned}
\widehat{W}(v_1,v_2) &\leq C(v_1 + t_2 - t_1 - \frac{\sum_{n\in Q} l_n^{max}}{C} && (31)\\
&\quad - \frac{\delta(C)}{C} - v_1) + \sum_{n\in Q} l_n^{max} \\
&\leq C(t_2 - t_1) - \delta(C) && (32)
\end{aligned}$$

Let $\widehat{t_2}$ be such that $v(\widehat{t_2}) = v_2$. Also, let $T(w)$ be the time taken by server to serve packets with aggregate length w in its busy period. Then,

$$\begin{aligned}
\widehat{t_2} &\leq t_1 + T(\widehat{W}(v_1,v_2)) && (33)\\
&\leq t_1 + T(C(t_2 - t_1) - \delta(C)) && (34)
\end{aligned}$$

From the definition of Fluctuation Constrained server, we get:

$$T(w) \leq \frac{w}{C} + \frac{\delta(C)}{C} \tag{35}$$

From (34) and (35) we get:

$$\begin{aligned}
\widehat{t_2} &\leq t_1 + \frac{C(t_2 - t_1) - \delta(C)}{C} + \frac{\delta(C)}{C} && (36)\\
&\leq t_2 && (37)
\end{aligned}$$

From Lemma 1 we know that:

$$W_f(t_1, \widehat{t_2}) \geq r_f(v_2 - v_1) - l_f^{max} \tag{38}$$

Since $\widehat{t_2} \leq t_2$, using (30) we get:

$$W_f(t_1,t_2) \geq r_f(t_2 - t_1) - r_f\frac{\sum_{n\in Q} l_n^{max}}{C} - r_f\frac{\delta(C)}{C} - l_f^{max} \tag{39}$$

■

Theorem 3 *If Q is the set of flows served by a SFQ EBF server with parameters $(C, B, \alpha, \delta(C))$, and $\sum_{n\in Q} r_n \leq C$, then for all intervals $[t_1, t_2]$ in which flow f is backlogged throughout the interval, $W_f(t_1, t_2)$ is given as:*

$$\begin{aligned}
P(W_f(t_1,t_2) &< r_f(t_2 - t_1) - r_f\frac{\sum_{n\in Q} l_n^{max}}{C} - r_f\frac{\delta(C)}{C} \\
&\quad - r_f\frac{\gamma}{C} - l_f^{max}) \leq Be^{-\alpha\gamma} \quad 0 \leq \gamma && (40)
\end{aligned}$$

The throughput guarantees of other fair scheduling algorithms have not been established. However, it can be shown that for constant rate servers, when similar admission control procedure is used, the throughput guarantee of neither WFQ nor SCFQ is better than that of SFQ.

Observe that throughput guarantees derived in Theorems 2 and 3 for SFQ have a recursive structure. Specifically, the throughput guaranteed to a flow by an FC or an EBF SFQ server is also fluctuation constrained or has exponentially bounded fluctuation, respectively. We will exploit this recursive structure to analyze performance bounds when a link bandwidth is hierarchically partitioned.

2.3 Delay Guarantee

SFQ algorithm, as defined so far, only allocates constant rate to the packets of a flow. However, due to the multiple time-scale variation of VBR video, to achieve efficient utilization of network resources, a server may be required to allocate variable rate to packets of a video flow. To support variable rate allocation, we generalize SFQ by extending the definition of service tags. Let r_f^j be the rate assigned to packet p_f^j. Then finish tag of packet p_f^j, $F(p_f^j)$ is defined as:

$$F(p_f^j) = S(p_f^j) + \frac{l_f^j}{r_f^j} \quad j \geq 1 \tag{41}$$

Start tag of a packet and the system virtual time are defined as before. We now prove the delay guarantee of the generalized SFQ algorithm.

Clearly, a server can provide a bound on delay only if its capacity is not exceeded. However, when variable rate is allocated to the packets of a flow, the intuitive meaning of capacity not being exceeded needs to be defined precisely. To do so, let rate function for flow f at virtual time v, denoted by $R_f(v)$, be defined as the rate assigned to the packet that has start tag less than v and finish tag greater than v. Formally,

$$R_f(v) = \begin{cases} r_f^j & \text{if } \exists j \ni \left(S(p_f^j) \leq v < F(p_f^j)\right) \\ 0 & \text{otherwise} \end{cases}$$

Let Q be the set of flows served by the server. Then a FC or EBF server with average rate C, is defined to have exceeded its capacity at virtual time v if $\sum_{n\in Q} R_n(v) > C$. If the capacity of a SFQ server is not exceeded, then it guarantees a deadline to a packet based on its *expected arrival time*. Expected arrival time of packet p_f^j that has been assigned rate r_f^j, denoted by $EAT(p_f^j, r_f^j)$, is defined as:

$$\max\{A(p_f^j), EAT(p_f^{j-1}, r_f^{j-1}) + \frac{l_f^{j-1}}{r_f^{j-1}}\} \quad j \geq 1 \tag{42}$$

where $EAT(p_f^0, r_f^0) = -\infty$. A deadline guarantee based on expected arrival time has been referred to as *delay guarantee* [8, 16]. Theorems 4 and 5 establish the delay guarantee of SFQ for FC and EBF servers, respectively.

Theorem 4 *If Q is the set of flows served by a SFQ FC server with parameters $(C, \delta(C))$ and $\sum_{n\in Q} R_n(v) \leq C$ for all v, then the departure time of packet p_f^j at the server, denoted by $L_{SFQ}(p_f^j)$, is given by:*

$$L_{SFQ}(p_f^j) \leq EAT(p_f^j, r_f^j) + \sum_{n\in Q \wedge n\neq f} \frac{l_n^{max}}{C} + \frac{l_f^j}{C} + \frac{\delta(C)}{C} \tag{43}$$

Proof: Let set H be defined as follows:

$$H = \{m | m > 0 \wedge S(p_f^m) = v(A(p_f^m))\} \tag{44}$$

Let $k \leq j$ be largest integer in H. Also, let $v_1 = v(A(p_f^k))$ and $v_2 = S(p_f^j)$. Observe that as the server virtual time is set to the maximum finish tag assigned to any packet at the end of a busy period, packet p_f^k and p_f^j are served in the same busy period of a server. From the definition of SFQ, the set of flow f packets served in the interval $[v_1, v_2]$ have start tag at least v_1 and at most v_2. Hence, the set can be partitioned into two sets:

- This set consists of packets that have start tag at least v_1 and finish tag at most v_2. Formally the set of packets of flow n, denoted by D_n, in this set is:

$$D_n = \{m | v_1 \leq S(p_n^m) \leq v_2 \wedge F(p_n^m) \leq v_2\} \tag{45}$$

Then, from the definition of $R_n(v)$ and $F(p_n^m, r_f^m)$, we know that the cumulative length of such flow n packets served by the server in the virtual time interval $[v_1, v_2]$, denoted by $AP_n(v_1, v_2)$, is given as:

$$AP_n(v_1, v_2) \leq \int_{v_1}^{v_2} R_n(v)dv \tag{46}$$

Hence, aggregate length of packets in this set, $\sum_{n \in Q} AP_n(v_1, v_2)$,is given as:

$$\begin{aligned} \sum_{n \in Q} AP_n(v_1, v_2) &\leq \sum_{n \in Q} \int_{v_1}^{v_2} R_n(v)dv & (47) \\ &\leq \int_{v_1}^{v_2} \sum_{n \in Q} R_n(v)dv & (48) \\ &\leq \int_{v_1}^{v_2} Cdv & (49) \\ &\leq C(v_2 - v_1) & (50) \end{aligned}$$

But since $v_2 = S(p_f^j, r_f^j)$, from the definition of k, $v_2 - v_1 = \sum_{n=0}^{n=j-k-1} \frac{l_f^{k+n}}{r_f^{k+n}}$. Hence,

$$\sum_{n \in Q} AP_n(v_1, v_2) \leq C \sum_{n=0}^{n=j-k-1} \frac{l_f^{k+n}}{r_f^{k+n}} \tag{51}$$

- This set consists of packets that have start tag at most v_2 and finish tag greater than v_2. Formally, the set of packets of flow n, denoted by E_n, in this set is

$$E_n = \{m | v_1 \leq S(p_n^m) \leq v_2 \wedge F(p_n^m) > v_2\} \tag{52}$$

Clearly, at most one packet of flow n can belong to this set. Furthermore, $E_f = \{j\}$. Hence, the maximum aggregate length of packets in this set is:

$$\sum_{n \in Q \wedge n \neq f} l_n^{max} + l_f^j \tag{53}$$

Hence, the aggregate length of packets served by the server in the interval $[v_1, v_2]$, denoted by $\widehat{W}(v_1, v_2)$, is:

$$\widehat{W}(v_1, v_2) \leq C \sum_{n=0}^{n=j-k-1} \frac{l_f^{k+n}}{r_f^{k+n}} + \sum_{n \in Q \wedge n \neq f} l_n^{max} + l_f^j \tag{54}$$

Let $T(w)$ be the time taken by server to serve packets with aggregate length w in a busy period. From the definition of Fluctuation Constrained server, we get:

$$T(w) \leq \frac{w}{C} + \frac{\delta(C)}{C} \tag{55}$$

Since packet p_f^j departs at system virtual time v_2 and all the packets served in the virtual time interval $[v_1, v_2]$ are served in the same busy period of the server, we get:

$$A(p_f^k) + T(\widehat{W}(v_1, v_2)) \geq L_{SFQ}(p_f^j) \tag{56}$$

$$\begin{aligned} A(p_f^k) &+ \textstyle\sum_{n=0}^{n=j-k-1} \frac{l_f^{k+n}}{r_f^{k+n}} + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} \\ &+ \frac{l_f^j}{C} + \frac{\delta(C)}{C} \geq L_{SFQ}(p_f^j) \qquad (57) \end{aligned}$$

From (42) we get

$$EAT(p_f^j, r_f^j) + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^j}{C} + \frac{\delta(C)}{C} \geq L_{SFQ}(p_f^j) \tag{58}$$

■

Theorem 5 *If Q is the set of flows served by a SFQ EBF server with parameters $(C, B, \alpha, \delta(C))$ and $\sum_{n \in Q} R_n(v) \leq C$ for all v, then the departure time of packet p_f^j at the server, denoted by $L_{SFQ}(p_f^j)$, is given by:*

$$\begin{aligned} P \quad (\quad & L_{SFQ}(p_f^j) \leq EAT(p_f^j, r_f^j) + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} \\ & + \frac{l_f^j}{C} + \frac{\delta(C)}{C} + \frac{\gamma}{C}) \geq 1 - Be^{-\alpha\gamma} \quad 0 \leq \gamma \qquad (59) \end{aligned}$$

The delay guarantee derived in Theorems 4 and 5[5] is independent of a tie breaking rule that a SFQ server may use when more than one packet have the same start tag. Though a tie breaking rule does not effect the delay guarantee, it can be used by a server to achieve different objectives. For example, a tie-breaking rule may give higher priority to interactive, low-throughput applications to reduce the average delay.

Theorems 4 and 5 can be used to determine delay guarantee even when a server has flows with different priorities and services them in the priority order (such a scenario may occur in an integrated services network with different traffic types). To illustrate, consider a server that services flows with two priorities and uses SFQ to schedule the packets of lower priority flows. If the bandwidth requirement of flows that are given higher priority can be characterized by a leaky bucket with average rate ρ and burstiness σ (such a characterization may be enforced by shaping the higher priority flows through a leaky bucket), then the residual bandwidth available to the lower priority flows can be modeled as fluctuation constrained with parameters $(C - \rho, \sigma)$. Hence, Theorem 4 can be used to determine the delay guarantee of the lower priority flows. Similarly, if the aggregate arrival process of the high priority flows can be modeled as poisson process, then the residual bandwidth can be modeled as EBF server [17] and Theorem 5 can be used to determine the delay guarantee.

[5] The proof methods of Theorem 4 and 5 can be used to derive delay guarantee of FC and EBF SCFQ servers.

Theorem 4 demonstrates that maximum delay of a packet in SFQ is significantly smaller than in SCFQ. Specifically, a tight bound on the departure time of a packet at a constant rate server employing SCFQ, given in [9], is:

$$L_{SCFQ}(p_f^j) \leq EAT(p_f^j, r_f^j) + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^j}{r_f^j} \qquad (60)$$

Since $\delta(C) = 0$ for a constant rate server, the difference in maximum delay that a packet may incur at servers employing SCFQ and SFQ is:

$$\frac{l_f^j}{r_f^j} - \frac{l_f^j}{C} \qquad (61)$$

Clearly, maximum delay in SFQ is much smaller than in SCFQ. To illustrate numerically, when r_f^j =64Kb/s, $l_f^j = 200$ bytes and C=100Mb/s, the difference is 24.4ms. If there are K servers on the path of a flow, this difference increases by a factor of K. For instance, the difference increases to 122ms for $K = 5$. Similarly, the difference increases linearly with the packet size.

Theorem 4 demonstrates that SFQ does not couple bandwidth and delay allocation. This enables it to provide lower maximum delay to low-throughput applications as compared to WFQ that inversely couples delay and bandwidth allocation. To observe this, consider the difference in the maximum delay experienced by packet p_f^j, denoted by $\Delta(p_f^j)$, in WFQ and SFQ. Since WFQ guarantees that packet p_f^j will be transmitted by $EAT(p_f^j, r_f^j) + \frac{l_f^j}{r_f^j} + \frac{l_{max}}{C}$ where l_{max} is the maximum packet length at the server, we get:

$$\Delta(p_f^j) = \frac{l_f^j}{r_f^j} + \frac{l_{max}}{C} - \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} - \frac{l_f^j}{C} \qquad (62)$$

To gain a qualitative understanding of $\Delta(p_f^j)$, let $l_f^j = l_{max} = l_n^{max} = l$ and $r_f^j = r_f$. Then,

$$\Delta(p_f^j) = \frac{l}{r_f} - \frac{(| Q | -1)l}{C} \qquad (63)$$

Hence, $\Delta(p_f^j) \geq 0$ if:

$$\frac{1}{| Q | -1} \geq \frac{r_f}{C} \qquad (64)$$

This shows that maximum delay of packets of a flow in SFQ is smaller than in WFQ if the fraction of the link bandwidth used by the flow is at most $\frac{1}{|Q|-1}$, which is expected to be true for low-throughput applications. This is also illustrated by Figure 3, which plots the reduction in delay in SFQ for different number of flows and flow rates, assuming 200 bytes packet and link capacity of 100Mb/s. As the figure shows, the reduction is higher for lower throughput flows. To compare the delay performance of WFQ and SFQ, consider a network link that is servicing 70 flows (possibly video flows) with rate 1Mb/s and 200 flows (possibly audio flows) with rate 64Kb/s. In such a scenario, whereas the maximum delay of the packets of flow with rate 64 Kb/s reduces by 20.39ms in SFQ, the maximum delay of 1Mb/s flows increases by 2.48 ms. Hence, SFQ reduces the maximum delay of low throughput flows significantly without an appreciable increase in the delay of high throughput flows.

SFQ not only reduces the maximum delay as compared to WFQ, but is also expected to lower the average delay of low-throughput applications. This is because whereas SFQ schedules packets in the increasing order of start tags, and thereby schedules packets at the earliest possible instant; WFQ schedules packets in increasing order of finish tag, and thus delays a packet as long as possible. To validate this hypothesis, we simulated a switch that was shared by high and low throughput flows carrying poisson traffic. The link capacity was 1Mb/s and the packet size was 200 bytes. Seven high-throughput flows with average rate 100Kb/s shared the switch with varying number of low-throughput flows with average rate 32Kb/s. The number of low-throughput flows was varied from 2 to 10 and the switch was simulated for 1000 seconds. Figure 4 compares the average packet delay of low-throughput flows in WFQ and SFQ at varying levels of link utilization. As the figure illustrates, the average delay of low-throughput flows is significantly higher in WFQ than in SFQ. In fact, at 80.81% link utilization, the average delay is 53% higher in WFQ than in SFQ. Hence, SFQ provides lower maximum as well as average delay to low-throughput applications as compared to WFQ.

As is evident from the definition of the expected arrival time, two key properties of the delay guarantee of SFQ for a flow are: (1) it is independent of the behavior of other sources at the server, and thereby isolates the flow; and (2) it is independent of a traffic characterization. Whereas the isolation property enables a server to provide stronger guarantees to the flow and is highly desirable when sources may be malicious [2, 5, 16, 19], independence of delay guarantee from traffic characterization enables a server to provide various QoS guarantees to flows conforming to any specification [9]. To enable a network of servers to provide similar guarantees, in what follows, we derive end-to-end delay guarantee.

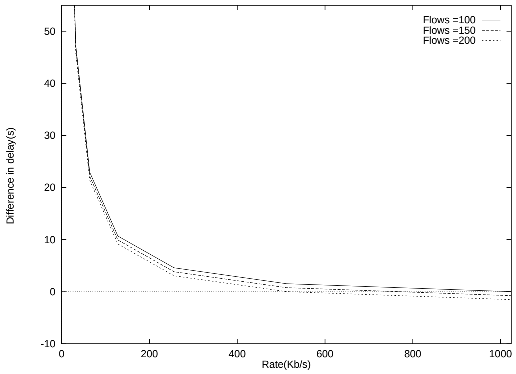

Figure 3 : Difference in maximum delay in WFQ and SFQ

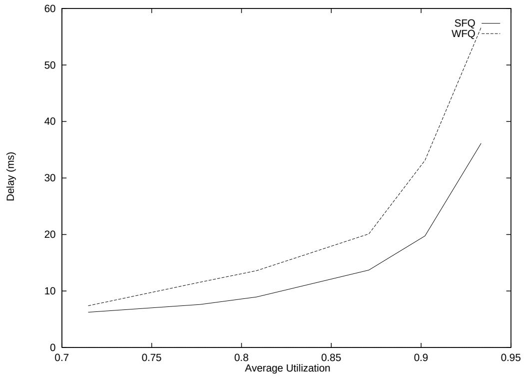

Figure 4 : Comparison of average delay in WFQ and SFQ

2.4 End-to-End Delay Guarantee

The objective is to determine the deadline guarantee of a network of servers based on the expected arrival time of a packet at the first server on the path of a flow [9]. To do so, let the i^{th} server along the path of a flow be denoted as server i. Also, let there be K servers on the path of a flow and let each of the server guarantee a deadline to a packet based on its expected arrival time. Then, the network guarantees a deadline to a packet based on its expected arrival time at the K^{th} server. Observe that the expected arrival time of a packet at server K is dependent on departure time of packet at server $K-1$, which, in turn, is dependent on expected arrival time of the packet at server $K-1$. Using this argument recursively, a network of servers can guarantee a deadline to a packet based on the expected arrival time of the packet at the first server. This method has been used in [9] to derive end-to-end delay guarantee of a network of servers that employ algorithms in the class of Guaranteed Rate (GR) scheduling algorithms. However, the end-to-end delay guarantee presented in [9] assumes that each of the server provides a deterministic bound on the departure time of a packet. Consequently, even though SFQ belongs to GR, the guarantee is not applicable to a network which may have some SFQ EBF servers. To analyze such networks, we generalize the method presented in [9].

Observe that SFQ delay guarantee for both FC and EBF servers can be rewritten as:

$$P\left(L_{SFQ}(p_f^j) \leq EAT(p_f^j, r_f^j) + \beta_f^j + \gamma\right) \geq 1 - Be^{-\lambda\gamma} \quad (65)$$

where $\gamma \geq 0$. Substituting $\beta_f^j = \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^j}{C} + \frac{\delta(C)}{C}$, $B = 0$ and $\lambda = \infty$, yields the delay guarantee for FC server. Substituting $\beta_f^j = \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^j}{C} + \frac{\delta(C)}{C}$ and $\lambda = \alpha C$, on the other hand, yields the delay guarantee for EBF server. Hence, we will use (65) to derive the end-to-end delay guarantee. Furthermore, to facilitate interoperability with other scheduling algorithms, we will only require each server on the path of a flow to guarantee a deadline which is similar to (65). We first relate the expected arrival time of a packet at adjacent servers in Theorem 6 and then use it to derive end-to-end delay guarantee in Corollary 1.

Let $\tau^{i,i+1}$ be an upper bound on the propagation delay between servers i and $i+1$. Also, let all the variables of server i be identified by superscript i, i.e., β_f^j and r_f^j are identified as $\beta_f^{j,i}$ and $r_f^{j,i}$, respectively. Henceforth in this section, we will refer to a single flow f, and, hence drop the subscript f from all the variables.

Theorem 6 *If scheduling algorithm at server i guarantees that:*

$$P\left(L^i(p^j) \leq EAT^i(p^j, r^{j,i}) + \beta^{j,i} + \gamma\right) \geq 1 - B^i e^{-\lambda^i\gamma} \quad (66)$$

where $L^i(p^j)$ is the time at which packet p^j departs server i, then:

$$P\left(EAT^{i+1}(p^j, \widehat{r}^{j,i}) \leq EAT^i(p^j, \widehat{r}^{j,i}) + \max_{n \in [1..j]}\{\beta^{n,i}\} + \tau^{i,i+1} + \gamma\right) \geq 1 - B^i e^{-\lambda^i\gamma}$$

where $\widehat{r}^{j,i} \leq \min\{r^{j,i}, r^{j,i+1}\}$ and $\gamma \geq 0$.

Corollary 1 *If scheduling algorithm at each server on the path of a flow satisfies (66), and there are K servers on the path of the flow, then:*

$$P\left(L^K(p^j) \leq EAT^1(p^j, \widehat{r}^j) + \sum_{n=1}^{n=K} \max_{m \in [1..j]}\{\beta^{m,n}\} + \sum_{n=1}^{n=K-1} \tau^{n,n+1} + \gamma\right) \geq 1 - \left(\sum_{n=1}^{n=K} B^n\right) e^{-\gamma \frac{1}{\sum_{n=1}^{n=K} \frac{1}{\lambda^n}}}$$

where $L^K(p^j)$ is the time at which packet p^j leaves server K, and $\widehat{r}^j = \min_{n \in [1..K]} \widehat{r}^{j,n}$.

As Corollary 1 illustrates, the end-to-end guarantee, like the single server guarantee, is based on the expected arrival time of a packet and hence is independent of the behavior of other flows and any particular flow characterization. Moreover, as illustrated in [9], it can be used to determine various QoS parameters like upper bound on end-to-end delay, packet loss probability and buffer requirement for *any* traffic specification. Hence, such a guarantee is highly desirable.

To derive Corollary 1, we have only required the scheduling algorithm at each server to satisfy (66). Hence, any scheduling algorithm that satisfies (66) (for example, Virtual Clock, WFQ, and SCFQ) can interoperate to provide end-to-end guarantee. Furthermore, Corollary 1 can be used for an internetwork of FC and EBF servers. Finally, the proof method of Theorem 6 and Corollary 1 can be used to derive end-to-end delay guarantee even when packet may be fragmented and reassembled in the network. Hence, SFQ can provide guarantees in heterogeneous internetworking environments.

2.5 Discussion

SFQ borrows the notion of "self-clocking", i.e., computing system virtual time based on a tag of a packet in service, from SCFQ to achieve efficiency. However, it provides significantly lower delay than SCFQ while achieving the same fairness measure. Hence, using fairness, throughput, delay, and computational complexity as performance metrics, we conclude that SFQ is strictly better than SCFQ.

SFQ is similar in spirit to FQS which also schedules packets in the increasing order of start tags. However, as FQS uses $v(t)$ as defined in (3), it is computationally expensive and fails to allocate bandwidth fairly over variable rate servers. Furthermore, as Theorem 1 shows, its fairness measure is no smaller than that of SFQ. Finally, since in FQS, all Q flows can become active simultaneously, and consequently Q packets can have the same start tag, the bound on the departure time of a packet in FQS cannot be smaller than in SFQ. Hence, SFQ has many advantages but no disadvantages over FQS.

A key advantage of SFQ over WFQ, in addition to lower implementation complexity, is that it achieves fairness over variable rate servers while WFQ does not. Moreover, it provides considerably lower maximum and average delay to low-throughput applications than WFQ. Since low-throughput applications like audio and telnet are more delay sensitive than high-throughput applications like video and ftp, this feature is highly desirable. In case delay guarantee of WFQ is required, SFQ can be combined with non work-conserving Virtual Clock to derive *Fair airport scheduling* algorithm that provides the delay guarantee of WFQ and efficiently achieves fairness even over variable rate servers [11]. Thus, since SFQ addresses the drawbacks of WFQ while achieving its delay guarantee if desired, it is better suited than WFQ for integrated services networks.

Finally, since SFQ provides a bound on maximum delay that does not depend on the weights of other flows and has a bounded deviation from an optimal fair scheduling algorithm, it has better fairness and delay properties than DRR.

To summarize, we have shown that SFQ: (1) achieves low average as well as maximum delay for low-throughput applications; (2) provides fairness, regardless of variation in a server rate; (3) has a fairness measure that is at least as good as that of all the known fair scheduling algorithms; and (4) is computationally efficient. In the next section, we show that it enables hierarchical link sharing, and thus meets all the requirements of a scheduling algorithm for integrated services networks.

3 Hierarchical Link Sharing

Hierarchical link sharing is an ideal mechanism for managing heterogeneity in integrated services networks [6, 20]. It can be used by a network to support services that provide heterogeneous QoS as well as multiple protocol families that support different traffic types and/or congestion control mechanisms. For example, a network can support hard and soft real-time, as well as best effort services by partitioning the link bandwidth between them as per the expected requirements of each of the service. To support high and low reliability soft real-time services, the bandwidth of soft real-time service may be further partitioned. Similarly, the bandwidth of the best effort services may be further partitioned between throughput intensive and interactive services.

A key advantage of hierarchical link sharing is that it provides isolation between different services while enabling similar services to share resources. Hence, incompatible congestion control algorithms can coexist while compatible algorithms reap the advantages of sharing. For example, while high and low reliability soft real-time services get the benefits of sharing, the hard real-time service is isolated from the overbooking that may occur in soft real-time services, and the congestion control algorithm that may be used by the best effort services. Hierarchical link sharing also facilitates use of different resource allocation methods for different services. This is desirable as hard real-time services may use a scheduling algorithm that performs well when there is no overbooking; soft real-time services may prefer to use a scheduling algorithm that provides QoS guarantees and/or minimizes deadline violations in presence of overbooking [1]; and best effort services may use a fair scheduler for throughput intensive flow-controlled data applications.

The requirements of hierarchical link sharing is specified by a tree, referred to as link-sharing structure, in which each node, other than possibly leaf nodes, denotes an aggregation of flows [6]. Each node in the tree is referred to as a class and has a weight associated with it. The objective of a mechanism implementing hierarchical link sharing is to distribute the bandwidth allocated to a class among its subclasses fairly, i.e., in proportion to the weights [20]. This objective can be achieved by a *hierarchical scheduler* that considers each class, other than the leaf classes, as a virtual server and uses a *fair* scheduler to schedule the virtual servers. However, as the following example illustrates, the scheduler used must allocate bandwidth fairly even over variable rate servers.

Example 3 *Consider a link sharing structure in which classes A and B are subclasses of the root class. Let classes C and D be subclasses of class A and let each class have weight 1. Initially, let there be no packets in class B. Hence, class A gets the full link bandwidth. When class B also becomes active, the bandwidth available to class A (and hence to subclasses C and D) reduces to 50% of the link bandwidth. Consequently, to fairly partition the bandwidth of class A between subclasses C and D, the scheduler must be able to allocate bandwidth fairly over variable rate servers.*

SFQ is the only scheduling algorithm that has been demonstrated to allocate bandwidth fairly even over variable rate servers. In what follows, we present a hierarchical SFQ scheduler for link sharing.

Hierarchical SFQ scheduler is simple. It uses SFQ to schedule each class; treating each subclass as a flow. The scheduling of packets occurs recursively: the scheduler for root class schedules the subclasses; the scheduler of subclasses in turn schedule their subclasses. Since SFQ fairly allocates bandwidth regardless of the server behavior, this simple recursive hierarchical scheduling ensures that bandwidth allocated to a class is fairly allocated between the subclasses and thereby achieves the objective of hierarchical link sharing. Moreover, in contrast to link sharing mechanism in [6], it provides bounds on various performance measures:

- **Throughput Guarantee**: Consider a class f that is a subclass of the root class. Let the link be FC server with parameters $(C, \delta(C))$ and let the set of the subclasses of the root class be denoted by Q. Then, if class f has been assigned rate r_f, from Theorem 2 we conclude that the virtual server corresponding to f is a FC server with parameters:

$$\left(r_f, r_f \frac{\sum_{n\in Q} l_n^{max}}{C} + r_f \frac{\delta(C)}{C} + l_f^{max}\right) \tag{67}$$

  Similarly, using Theorem 3, we conclude that if the link is an EBF server, then the virtual server corresponding to f is a EBF server. Hence, if the link is an FC or EBF server, then the virtual servers corresponding to the subclasses of the root class are FC or EBF servers, respectively. Using the argument recursively, we conclude that if the link is a FC or EBF server, then each of the virtual server in the hierarchical structure is a FC or EBF server, respectively. Consequently, Theorems 2 and 3 can be used to determine the throughput guarantee of the flows.

- **Delay Guarantee**: Since each of the virtual server is either FC or EBF server, Theorems 4 and 5 can be used to determine the single server delay guarantee of the flows.

- **End-to-End Delay Guarantee**: Since the single server delay guarantee when a flow is hierarchically scheduled satisfies (66), Corollary 1 can be used to determine the end-to-end delay guarantee.

The elegance of the above analysis is in its simplicity. This simplicity demonstrates the generality of the analysis of SFQ servers presented in Section 2[6].

Hierarchical SFQ scheduler not only achieves the objectives of hierarchical link sharing, but can also be used to achieve several other objectives. For example it can be used to achieve:

- *Separation of delay and throughput allocation*: Observe that SFQ does not allocate delay and throughput separately. However, it may be desirable to do so for some flows. This can be achieved by aggregating the flows for which separation of delay and throughput is desirable into one class and then using a scheduling algorithm that achieves such a separation for that class. Though conceptually simple, since the throughput of a class fluctuates over time, the algorithm used must be able to achieve the separation over variable rate servers. In Theorem 7, we show that Delay EDD can achieve this over a FC server. Since the throughput of a class is fluctuation constrained, Delay EDD can be used to achieve the objective.

  We first define Delay EDD and then prove its delay guarantee for a FC server. On arrival of packet p_f^j of flow f, Delay EDD assigns it a deadline, denoted by $D(p_f^j)$, and schedules packets in increasing order of deadline. $D(p_f^j)$ is defined as:

$$D(p_f^j) = EAT(p_f^j, r_f) + d_f \tag{68}$$

  where d_f is the deadline of flow f packets, $r_f = r_f^j$, and $l_f = l_f^j$.

[6] We would like to warn the reader of a potential pitfall. It is possible that some may reach the conclusion that this analysis would hold even if the throughput of a server was modeled by a Service Burstiness server which is similar to a FC server [3]. However, even though the throughput guaranteed to a flow by a Virtual Clock server conforms to Service Burstiness, it can be shown that Virtual Clock when used for hierarchical link sharing provides no guarantees.

Theorem 7 *If Q is the set of flows serviced by the server and*

$$\forall t > 0: \quad \sum_{n \in Q} \max\{0, \left\lceil \frac{(t - d_n) r_n}{l_n} \right\rceil \frac{l_n}{C}\} \leq t \qquad (69)$$

and the server is a $(C, \delta(C))$ Fluctuation Constrained Delay EDD server, then the time at which the transmission of packet p_f^j is completed, denoted by $L_{EDD}(p_f^j)$, is:

$$L_{EDD}(p_f^j) \leq D(p_f^j) + \frac{l_{max}}{C} + \frac{\delta(C)}{C} \qquad (70)$$

Due to high computational complexity, it may not be feasible to employ (69) as the schedulability test. Hence, conditions stronger than (69) which have lower computational complexity have been developed in [25]. The theorem holds under the stronger conditions as well.

- *Delay shifting:* This involves the reduction of the maximum delay of certain flows at the expense of increasing the delay of others. To illustrate, let the server be a FC server with parameters $(C, \delta(C))$ and let the set of flows served by it be denoted by Q. For ease of exposition, let the packet length of each flow be l. Hence, using Theorem 4, for packet p_f^j:

$$L_{SFQ}(p_f^j) \leq EAT(p_f^j, r_f) + \frac{(|Q|-1)l}{C} + \frac{\delta(C)}{C} + \frac{l}{C} \qquad (71)$$

Now, let the set Q be partitioned into K sets $Q_1, \ldots, Q_K$ and let them be hierarchically scheduled. Let flow f belong to partition Q_i and the rate assigned to partition Q_n be denoted by C_n. To determine delay guarantee of flow f when it is hierarchically scheduled, observe from (67) that the virtual server corresponding to partition Q_i is a FC server with parameters $(C_i, \frac{C_i(\delta(C)+Kl)}{C} + l)$. Hence, the departure time of packet p_f^j of flow f, denoted by $\widehat{L_{SFQ}}(p_f^j)$, is given as:

$$\begin{aligned} \widehat{L_{SFQ}}(p_f^j) \quad \leq \quad & EAT(p_f^j, r_f) + \frac{(|Q_i|+1)l}{C_i} \\ & + \frac{\delta(C)+Kl}{C} \end{aligned} \qquad (72)$$

From (71) and (72), we conclude that the bound on the departure time would be smaller when the flow is hierarchically scheduled if:

$$\frac{(|Q_i|+1)l}{C_i} + \frac{\delta(C)+Kl}{C} - \frac{|Q|l}{C} - \frac{\delta(C)}{C} < 0 \qquad (73)$$

$$\Rightarrow \frac{|Q_i|+1}{|Q|-K} < \frac{C_i}{C} \qquad (74)$$

Hence, by ensuring that (74) is satisfied for the partitions which require lower delay, delay shifting can be achieved.

4 Implementation

We have implemented SFQ scheduler for a FORE Systems ATM network interface in Solaris 2.4 as a streams module and driver (see Figure 5). The driver is used to maintain a hierarchical link sharing structure, created via ioctl() calls, for a network interface. The module, on the other hand, is used to schedule packets. We have modified the FORE API for opening a connection to include the weight of a connection and its class as parameters.

To experimentally validate the implementation of the scheduler, we initiated three connections with weights 1, 2, and 3. Each of the connection terminated after transmitting 500,000, 4KB packets. Figure 5 shows the throughput received by each connection. As it demonstrates, when all the three connections were active, they received throughput in the ratio 1:2:3. When the connection with weight 3 terminated, the throughput of the other two connections increased but still remained in the ratio 1:2. Finally, when only one connection remained, it received the full link bandwidth. The throughput of the interface in this experiment was 48Mb/s which was the same without the SFQ scheduler (i.e., the scheduler did not impose any overhead). Observe from Figure 5 that SFQ scheduler achieved fair allocation even though the realizable bandwidth of the interface varied over time. This demonstrates the feasibility of employing SFQ for scheduling network interface in operating systems where the processing capacity available for a network interface varies over time.

5 Concluding Remarks

In this paper, we presented Start-time Fair Queuing (SFQ) algorithm that is computationally efficient, achieves fairness regardless of variation in a server capacity, and has the smallest fairness measure among all known fair scheduling algorithms. We analyzed its throughput, single server delay, and end-to-end delay guarantee for variable rate Fluctuation Constrained (FC) and Exponentially Bounded Fluctuation (EBF) servers. This is the first analysis of any fair or real-time scheduling algorithm for such servers. Our analysis demonstrated that SFQ is better suited than WFQ for integrated services networks and it is strictly better than SCFQ and FQS. To support heterogeneous services and multiple protocol families in integrated services networks, we presented a hierarchical SFQ scheduler. We derived performance bounds for flows that are hierarchically scheduled using a conceptually simple and elegant method. Finally, we presented an implementation of SFQ scheduler and demonstrated that it achieves fair allocation of bandwidth.

In summary, we demonstrated that SFQ: (1) achieves low average as well as maximum delay for low throughput applications (e.g., interactive audio, telnet, etc.); (2) provides fairness which is desirable for VBR video; (3) provides fairness, regardless of variation in server capacity, for throughput-intensive, flow-controlled data applications; (4) enables hierarchical link sharing which is desirable for managing heterogeneity; and (5) is computationally efficient. Thus, SFQ meets the requirements of a suitable scheduling algorithm for integrated services networks.

References

[1] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha. On-line Scheduling in the Presence of Overload. In *Proceeding of Foundations of Computer Science*, pages 100–110, 1991.

[2] D.D. Clark, S. Shenker, and L. Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network. In *Proceedings of ACM SIGCOMM*, pages 14–26, August 1992.

[3] R.L. Cruz. Service Burstiness and Dynamic Burstiness Measures: A Framework. *Journal of High Speed Networks*, 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.

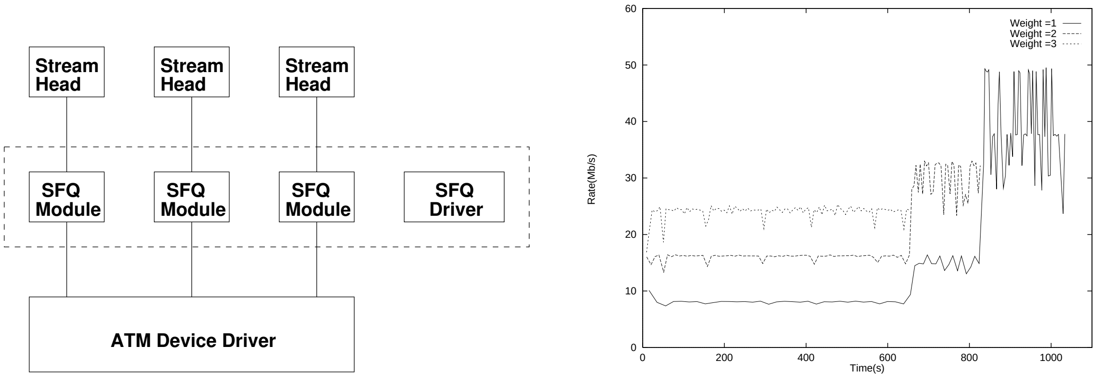

Figure 5 : (a) SFQ scheduler implementation (b) Throughput of the connections

[5] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *Proceedings of ACM SIGCOMM*, pages 1–12, September 1989.

[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.J. Golestani. A Self-Clocked Fair Queueing Scheme for High Speed Applications. In *Proceedings of INFOCOM'94*, 1994.

[8] P. Goyal, S. S. Lam, and H. M. Vin. Determining End-to-End Delay Bounds In Heterogeneous Networks. In *ACM/Springer-Verlag Multimedia Systems Journal (to appear)*, 1996. Also appeared in the Proceedings of the Workshop on Network and Operating System Support for Digital Audio and Video, April 1995.

[9] P. Goyal and H. M. Vin. Generalized Guaranteed Rate Scheduling Algorithms: A Framework. Technical Report TR-95-30, The University of Texas at Austin, July 1995.

[10] P. Goyal and H. M. Vin. Network Algorithms and Protocol for Multimedia Servers. In *Proceedings of INFOCOM'96*, pages 1371–1379, March 1996.

[11] P. Goyal, H. M. Vin, and H. Cheng. Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks. Technical Report TR-96-02, The University of Texas at Austin, January 1996. Available via URL http://www.cs.utexas.edu/users/dmcl.

[12] P. Goyal, H. M. Vin, C. Shen, and P.J. Shenoy. A Reliable, Adaptive Protocol for Video Transport. In *Proceedings of INFOCOM'96*, pages 1080–1090, March 1996.

[13] A. Greenberg and N. Madras. How Fair is Fair Queuing. *The Journal of ACM*, 39(3):568–598, July 1992.

[14] M. Grossglauser, S. Keshav, and D. Tse. RCBR: A Simple and Efficient Service for Multiple Time-Scale Traffic. In *Proceedings of ACM SIGCOMM'95*, 1995.

[15] S. Keshav. A Control-Theoretic Approach to Flow Control. In *Proceedings of ACM SIGCOMM'91*, pages 3–15, 1991.

[16] S.S. Lam and G.G. Xie. Burst Scheduling: Architecture and Algorithm for Switching Packet Video. In *Proceedings of INFOCOM'95*, April 1995.

[17] K. Lee. Performance Bounds in Communication Networks With Variable-Rate Links. In *Proceedings of ACM SIGCOMM'95*, pages 126–136, 1995.

[18] A.K. Parekh. *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks*. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, 1992.

[19] S. Shenker. Making Greed Work in Networks: A Game-Theoretic Analysis of Switch Service Disciplines. In *Proceedings of ACM SIGCOMM'94*, pages 47–57, 1994.

[20] S. Shenker, L. Zhang, and D. Clark. A Scheduling Service Model and a Scheduling Architecture for an Integrated Services Packet Networks. Available via anonymous ftp from ftp://ftp.parc.xerox.com/pub/archfin.ps, 1995.

[21] M. Shreedhar and G. Varghese. Efficient Fair Queuing Using Deficit Round Robin. In *Proceedings of ACM SIGCOMM'95*, pages 231–242, 1995.

[22] D. Stiliadis and A. Varma. Design and Analysis of Frame-based Fair Queueing: A New Traffic Scheduling Algorithm for Packet Switched Networks. In *Proceedings of SIGMETRICS'96 (to appear)*, 1996.

[23] H. Zhang and S. Keshav. Comparison of Rate-Based Service Disciplines. In *Proceedings of ACM SIGCOMM*, pages 113–121, August 1991.

[24] L. Zhang. VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks. In *Proceedings of ACM SIGCOMM'90*, pages 19–29, August 1990.

[25] Q. Zheng and K. Shin. On the Abilitiy of Establishing Real-Time Channels in Point-to-Point Packet-switching Networks. *IEEE Transactions on Communications*, 42(3):1096–1105, March 1994.