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

Pawan Goyal, Harrick M. Vin, and Haichen Cheng

Abstract— **We present Start-time Fair Queuing (SFQ) algorithm that is computationally efficient and achieves fairness regardless of variation in a server capacity. We analyze its single server and end-to-end deadline guarantee for variable rate Fluctuation Constrained (FC) and Exponentially Bounded Fluctuation (EBF) servers. 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 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.

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 [11]. 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.
  - 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 [21], Delay EDD [5], etc., penalize channels for the use of idle bandwidth and do not provide bandwidth allocation guarantee in the presence of congestion [17]. Fair scheduling algorithms, on the other hand, guarantee that, regardless of prior usage or congestion, bandwidth would be allocated fairly[17]. 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 [4, 15]. 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 mul-

The authors are with the Distributed Multimedia Computing Laboratory, Department of Computer Sciences, The University of Texas at Austin, Taylor Hall 2.124, Austin, TX 78712. Their e-mail addresses are {pawang,vin,hccheng}@cs.utexas.edu, and their URL is http://www.cs.utexas.edu/users/dmcl.

tiple protocol families, the scheduling algorithm should facilitate hierarchical link sharing [6, 18]. Finally, to facilitate its implementation in high-speed networks, it should be computationally efficient. A scheduling algorithm that achieves 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* [21]. 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 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 ϕ_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, $\frac{W_f(t_1,t_2)}{\phi_f} - \frac{W_m(t_1,t_2)}{\phi_m} = 0$. 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 $\mid \frac{W_f(t_1,t_2)}{\phi_f} - \frac{W_m(t_1,t_2)}{\phi_m} \mid$ is as close to 0 as possible. However, it has been shown in [8] that if a packet scheduling algorithm guarantees that $\mid \frac{W_f(t_1,t_2)}{\phi_f} - \frac{W_m(t_1,t_2)}{\phi_m} \mid \leq H(f,m)$ for all intervals $[t_1, t_2]$ then $H(f,m) \geq \frac{1}{2}\left(\frac{l_f^{max}}{\phi_f} + \frac{l_m^{max}}{\phi_m}\right)$, 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. The function $H(f,m)$ is referred to as fairness measure.

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) [4] (also referred to as Packet-by-Packet Generalized Processor Sharing (PGPS) [17]). 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:

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

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

where $F(p_f^0) = 0$ and $v(t)$ is defined as:

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

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.

The 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 may require processing of $O(Q)$ events in a single packet transmission time, where Q is the number of flows served, and thus is considered computationally expensive [8]. Furthermore, to retain fairness when server rate varies over time, the definition of virtual time will have to be modified. The following examples illustrate that if the definition of virtual time is not modified and is based on the assumption that the capacity of a server is constant, then WFQ becomes unfair over variable rate servers.

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.

A similar example can be constructed for the case when the the actual capacity of the server is higher than the assumed capacity. Thus, we conclude that to ensure fairness over variable rate servers, the definition of system virtual time should be modified to depend on the time varying server capacity. This can be achieved by defining $v(t)$ as $\frac{dv(t)}{dt} = \frac{C(t)}{\sum_{j \in B(t)} \phi_j}$ where $C(t)$ is the capacity of server at time t. Without a priori knowledge of $C(t)$, computing $v(t)$ based on the new definition requires counting the number of bits transmitted by the server during various intervals as well as continuous evaluation of $v(t)$. The complexity of these operations makes the cost of computing $v(t)$ prohibitive. Thus, we conclude that: (1) if constant rate approximations are employed in WFQ for variable rate servers, then WFQ is unfair, and (2) modified WFQ algorithm that may retain fairness over variable rate servers is computationally prohibitive. Thus, WFQ is unsuitable for achieving fairness over variable rate servers. As we will outline in Section 3, to be useful for hierarchical link sharing [6, 18], a scheduling algorithm must provide fairness over variable rate servers. Consequently, WFQ is unsuitable for supporting hierarchical link sharing also.

Fair Queuing based on Start-time (FQS), proposed in [14], 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 disadvantages similar to that of WFQ.

Self Clocked Fair Queuing (SCFQ), originally proposed in [3] and later analyzed in [8], 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 $(\frac{l_f^{max}}{\phi_f} + \frac{l_m^{max}}{\phi_m})$, which is only a factor of two away from the lower bound [8]. 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 [10]. This may be unacceptably large in many cases.

Frame-based Fair Queuing (FFQ) was designed to retain the efficiency of SCFQ in computing the start and finish tags but ensure that the worst-case delay that can be guaranteed to a packet is the same as in WFQ [20]. The main limitation of FFQ is that due to its assumption of constant rate servers, it is unfair over variable rate servers. Furthermore, its $H(f,m)$ value depends on the minimum rate allocated by a server, and can deviate significantly from the lower bound.

Worst-case-fair weighted Fair Queuing (WF^{2}Q), proposed in [2], was designed to improve WFQ's emulation of hypothetical bit-by-bit round robin server. To achieve this objective, WF^{2}Q : (1) utilizes $v(t)$ as defined in (3) and computes start and finish tags as in WFQ, (2) defines a packet to be eligible at time t only if its start tag is at most $v(t)$, and (3) schedules eligible packets in the increasing order of finish tags. It has been shown that WF^{2}Q emulates the hypothetical server well and has a $H(f,m)$ value of $(\frac{l_f^{max}}{\phi_f} + \frac{l_m^{max}}{\phi_m})$ [2]. However, since it utilizes $v(t)$ as defined in (3), it is computationally inefficient and unsuitable for achieving fairness over variable rate servers.

WF^{2}Q+ has been recently, independent of our work, proposed to reduce the implementation complexity of WF^{2}Q while retaining several of its properties (a similar, but not identical, algorithm termed Starting Potential based Fair Queuing was proposed in [20]) [1]. It defines start tag of packet p_f^j to be the finish tag of packet p_f^{j-1}, i.e., $S(p_f^j) = F(p_f^{j-1})$, if flow f is backlogged on arrival of p_f^j; otherwise, $S(p_f^j) = \max\{v(A(p_f^j)), F(p_f^{j-1})\}$. The finish tag of a packet and the set of eligible packets are defined as in WF^{2}Q but $v(t)$ is defined as $v(t) = \max\{v(\tau) + t - \tau, \min_{n \in B(t)} S(p_n^k)\}$ where τ is the largest time less than t at which a packet finished service, p_n^k is the packet at the head of the queue of flow n at time t, and $B(t)$ is the set of backlogged flows at time t. WF^{2}Q+ , like WF^{2}Q , schedules eligible packets in increasing order of finish tags. Though *worst-case fairness* of WF^{2}Q+ has been derived, its fairness measure has not been derived in[1] [1]. To ensure that properties of WF^{2}Q+ hold over variable rate servers, it has been proposed in [1] that *reference time*, instead of real time, should be used in virtual time computation. Reference time at real time t, $T_R(t)$, is defined as $T_R(t) = \frac{W(0,t)}{C}$ where C is capacity of the server and $W(0,t)$ is the work done by the server in interval $[0,t]$. Given no a priori information regarding variation in server capacity, it appears that determining $W(0,t)$ will require counting the number of bits that have been transmitted by the server in the interval $[0,t]$; this computation can be expensive. Furthermore, WF^{2}Q+ has been studied under the assumption that $\sum_{n \in Q} \phi_n \leq C$ where C is the minimum capacity of a server. The following example demonstrates that this assumption is necessary to ensure fairness of WF^{2}Q+ .

Example 2 Let a server serve packets at a constant rate of $K+1 pkt/s$ in $[0,1]$ and then at the constant rate of $2pkt/s$. Thus, C is 2 pkt/s. Let the server serve $K+2$ flows and let each flow be assigned a weight of $1pkt/s$. Let flows $1..K$ terminate after sending one packet each at time 0 and let flow $K+1$ send infinite number of packets. Also let flow $K+2$ send one packet at time $t=1$. Now for all $n \in [1..K]$, $F(p_n^1) = 1$. The finish tags of flow $K+1$ packets are given as $F(p_{K+1}^j) = j$. Since $K+1$ packets are served by time 1, $v(1) = T_R(1) = \frac{K+1}{2}$. Thus, $F(p_{K+2}^1) = \frac{K+1}{2} + 1$. Since the first packet of flows $1..K+1$ are eligible at time 0 and WF^{2}Q+ schedules packets in the increasing order of finish tags, first packet of flows $1..K+1$ will be served in the time interval $[0,1]$. For ease of exposition of the later part of the schedule, let $q = \lceil \frac{K+1}{2} \rceil$. Then, since $S(p_{K+1}^q) = q-1$ and $v(1) > q-1$, packets $p_{K+1}^2, ..p_{K+1}^q$ are eligible for scheduling at time 1. Furthermore, since $F(p_{K+1}^q) < F(p_{K+2}^1)$, in the interval $[1, 1+\frac{q-1}{2}]$, $q-1$ packets of flow $K+1$ will be scheduled. Thus in the interval $[1, 1+\frac{q-1}{2}]$ even though flows $K+1$ and $K+2$ are backlogged, whereas $q-1$ packets of flow $K+1$ are served, no packet of flow $K+2$ is served. By choosing K, and hence q, appropriately, the difference in the service received by flows $K+1$ and $K+2$ can be made arbitrarily large.

$\sum \phi_i \leq C$ may be ensured either by dynamically changing the weight assignments of flows or by performing admission control. An algorithm for dynamically changing the weights or an evaluation of its effects on the fairness properties have not been presented in [1]. On the other hand, it

[1] An algorithm that has bounded worst-case-fairness may have unbounded fairness measure [12].

may not be possible to perform admission control for some flow types (for example, best-effort flows). Furthermore, it may not be feasible to employ admission control when minimum server capacity is zero.

WFQ, FQS, SCFQ, FFQ, WF^2Q , and WF^2Q+ 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 [19]. 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, its fairness measure can deviate arbitrarily from the lower bound. Furthermore, the maximum delay incurred by packets can be significantly higher than in WFQ [12].

In summary, the design of a fair scheduling algorithm that is: (1) computationally efficient, (2) provides fairness regardless of variation in server capacity, (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 admission control as well as variation in a server rate. We show that it has a fairness measure of $(\frac{l_f^{max}}{\phi_f}+\frac{l_m^{max}}{\phi_m})$, which, on an average, is 11% away from the tighter lower bound that we derive. We analyze the single server and end-to-end deadline 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 [16]. 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 single server and end-to-end deadline guarantees of a flow when the link bandwidth is hierarchically partitioned. We demonstrate that the hierarchical SFQ scheduler, in addition to supporting heterogeneity, can be used to achieve separation of delay and throughput allocation.

The rest of the paper is structured as follows. We present SFQ algorithm and analyze its fairness, throughput, single server deadline guarantee, and end-to-end deadline 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 algorithm (SFQ) 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:
$$S(p_f^j) = \max\{v(A(p_f^j)), F(p_f^{j-1})\} \quad j \geq 1 \qquad (4)$$
where $F(p_f^j)$, the finish tag of packet p_f^j, is defined as:
$$F(p_f^j) = S(p_f^j) + \frac{l_f^j}{\phi_f} \quad j \geq 1 \qquad (5)$$
where $F(p_f^0) = 0$ and ϕ_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. 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 then[2].
3. Packets are serviced in the increasing order of the start tags; ties are broken arbitrarily.

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 [16][3]. A FC server has two parameters; average rate C (bits/s) and burstiness $\delta(C)$ (s). Intuitively, in

[2]Observe that server virtual time changes only when a packet finishes service. Also, we set $v(t)$ to the maximum of the 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.

[3]The definitions of FC and EBF servers as presented here are different from that in [16]. Specifically, whereas [16] characterizes the servers by

an FC server, the time taken to serve packets of aggregate length w in a busy period can exceed the time taken in an equivalent constant rate server by at most $\delta(C)$. Formally,

Definition 1 A server is a Fluctuation Constrained (FC) server with parameters $(C, \delta(C))$, if the time taken to serve packets of aggregate length w in a busy period, denoted by $T(w)$, satisfies:

$$T(w) \leq \frac{w}{C} + \delta(C) \quad (6)$$

EBF server is a stochastic relaxation of FC server. Intuitively, in an EBF server, the probability of the time taken to serve packets of aggregate length w in a busy period deviating by more than γ from that in an equivalent constant rate server, decreases exponentially with γ. Formally,

Definition 2 A server is an Exponentially Bounded Fluctuation (EBF) server with parameters $(C, B, \alpha, \delta(C))$, if the time taken to serve packets of aggregate length w in a busy period, denoted by random variable $T(w)$, satisfies:

$$P(T(w) > \frac{w}{C} + \delta(C) + \gamma) \leq Be^{-\alpha\gamma} \quad 0 \leq \gamma \quad (7)$$

In what follows, we analyze the fairness of SFQ for any variable rate server, and its deadline 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. Due to space constraints, we omit the proofs and present them in [12].

2.1 Fairness Guarantee

To derive fairness guarantee of SFQ, we need to prove a bound on $\mid \frac{W_f(t_1,t_2)}{\phi_f} - \frac{W_m(t_1,t_2)}{\phi_m} \mid$ 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:

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

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

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

$$W_f(t_1, t_2) \leq \phi_f(v_2 - v_1) + l_f^{max} \quad (9)$$

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

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:

$$\mid \frac{W_f(t_1, t_2)}{\phi_f} - \frac{W_m(t_1, t_2)}{\phi_m} \mid \leq \frac{l_f^{max}}{\phi_f} + \frac{l_m^{max}}{\phi_m} \quad (10)$$

the work done in a busy period, we characterize the servers by the time taken to serve packets of length w in a busy period.

Theorem 1 demonstrates that SFQ has a $H(f, m)$ value of $\frac{l_f^{max}}{\phi_f} + \frac{l_m^{max}}{\phi_m}$. To evaluate the fairness guarantee of SFQ, we have derived a lower bound on $H(f, m)$ that is tighter than $\frac{1}{2}(\frac{l_f^{max}}{\phi_f} + \frac{l_m^{max}}{\phi_m})$ which was presented in [8]. Specifically, in [12], we have shown that $H(f, m) \geq L(f, m)$ where $L(f, m)$ is $(c+1)\alpha$, $\alpha = \frac{l_f^{max}}{\phi_f}$, $\beta = \frac{l_m^{max}}{\phi_m}$, $\alpha \leq \beta$, and c is a positive integer such that $c\alpha < \beta \leq (c+1)\alpha$. The fairness guarantee of SFQ, on an average, is within 11% of $L(f, m)$.

There are two important aspects of Theorem 1:

- To establish it, we did not make any assumptions about the service rate of the server. Hence, it 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.
- To establish it, we did not make any assumptions about the weights; weights are just uninterpreted numbers. In particular, we did not require any admission control such as $\sum_{n \in Q} \phi_n \leq C$. Since for variable rate servers, C may not always be defined, as well as it may not be possible to perform admission control for best-effort flows, this property is desirable. This is an important difference between SFQ and algorithms such as WF^{2}Q+ and FFQ.

2.2 Deadline Guarantee

In the previous sections, we have not assigned any interpretation to the weight of a flow. To establish the deadline guarantee of a flow, we will henceforth interpret ϕ_f as the rate assigned to flow f and denote ϕ_f by r_f. 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 the 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:

$$F(p_f^j) = S(p_f^j) + \frac{l_f^j}{r_f^j} \quad j \geq 1 \quad (11)$$

Start tag of a packet and the system virtual time are defined as before.

We show in Sections 2.2.1 and 2.2.2 that the generalized SFQ algorithm provides two types of deadline guarantees to a packet:

- It guarantees a deadline to a packet based on its *expected arrival time*. Specifically, it guarantees that:

$$L_{SFQ}(p_f^j) \leq EAT(p_f^j, r_f^j) + \beta_f^j \quad (12)$$

where $L_{SFQ}(p_f^j)$ is the departure time of packet p_f^j in a SFQ server, β_f^j depends on l_f^j and the properties of the server as well as the other flows at the server, and $EAT(p_f^j, r_f^j)$ is the expected arrival time of packet p_f^j that has been assigned rate r_f^j. $EAT(p_f^j, r_f^j)$ is defined as:

$$EAT(p_f^j, r_f^j) = \max\{A(p_f^j), EAT(p_f^{j-1}, r_f^{j-1}) + \frac{l_f^{j-1}}{r_f^{j-1}}\} \tag{13}$$

where $EAT(p_f^0, r_f^0) = -\infty$. Such a guarantee has been referred to as *delay guarantee* and is used to provide various QoS guarantees regardless of the behaviour of the other flows in the network [10].

- It guarantees a deadline to a packet based on its arrival time and the departure time of the previous packet. Specifically, it guarantees that:

$$L_{SFQ}(p_f^j) \leq \max\{L_{SFQ}(p_f^{j-1}), A(p_f^j)\} + \beta_f^j \tag{14}$$

Such a deadline guarantee, which we refer to *delay-cum-throughput guarantee*, improves upon the performance bounds determined from delay guarantee when the actual service received by a flow is better than that guaranteed by the server.

SFQ provides these deadline guarantees when the the server capacity is not exceeded. To derive the deadline guarantee, let us formalize the meaning of the term "capacity is not exceeded". 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 the capacity of a FC or EBF server with average rate C, is not exceeded if:

$$\sum_{n \in Q} R_n(v) \leq C \quad v \geq 0 \tag{15}$$

To derive the delay as well as delay-cum-throughput guarantee of FC and EBF SFQ servers, we first derive a bound on the work done by a SFQ server in virtual time interval $[v_1, v_2]$ in Lemma 3.

Lemma 3 If the capacity of a FC or EBF server with parameters $(C, \delta(C))$ or $(C, B, \alpha, \delta(C))$, respectively, is not exceeded, then the aggregate length of packets that have start tag at least v_1 and at most v_2 and are served in the same busy period, denoted by $\widehat{W}(v_1, v_2)$, is given by:

$$\widehat{W}(v_1, v_2) \leq C \sum_{n=0}^{n=j-k-1} \frac{l_f^{k+n}}{r_f^{k+n}} + \sum_{n \in Q \wedge n \neq f} l_n^{max} + l_f^j \tag{16}$$

whenever $v_1 = S(p_f^k, r_f^k)$, $v_2 = S(p_f^j, r_f^j)$ and $v_2 - v_1 = \sum_{n=0}^{n=j-k-1} \frac{l_f^{k+n}}{r_f^{k+n}}$.

For brevity, we will denote $\sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^j}{C} + \delta(C)$ by θ_f^j.

2.2.1 Delay Guarantee

Theorems 2 and 3 establish the delay guarantee of SFQ for FC and EBF servers, respectively.

Theorem 2 If the capacity of a SFQ FC server with parameters $(C, \delta(C))$ is not exceeded, then:

$$L_{SFQ}(p_f^j) \leq EAT(p_f^j, r_f^j) + \theta_f^j \tag{17}$$

Theorem 3 If the capacity of a SFQ EBF server with parameters $(C, B, \alpha, \delta(C))$ is not exceeded, then:

$$P\left(L_{SFQ}(p_f^j) \leq EAT(p_f^j, r_f^j) + \theta_f^j + \gamma\right) \geq 1 - Be^{-\alpha\gamma} \tag{18}$$

The delay guarantee derived in Theorems 2 and 3 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 2 and 3 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).

Theorem 2 demonstrates that maximum delay of a packet in SFQ is smaller than in SCFQ. Specifically, a tight bound on the departure time of a packet at a constant rate server employing SCFQ, given in [10], is:

$$L_{SCFQ}(p_f^j) \leq EAT(p_f^j, r_f^j) + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^j}{r_f^j} \tag{19}$$

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:

$$\frac{l_f^j}{r_f^j} - \frac{l_f^j}{C} \tag{20}$$

Clearly, maximum delay in SFQ is 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. Similarly, the difference increases linearly with the packet size.

Theorem 2 also shows that, unlike WFQ, the maximum delay of a packet in SFQ depends on the maximum packet length of all the flows at the server. However, in spite of this dependence, SFQ provides lower maximum delay, as compared to WFQ, to low-throughput flows. 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:

$$\Delta(p_f^j) = \frac{l_f^j}{r_f^j} + \frac{l_{max}}{C} - \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} - \frac{l_f^j}{C} \quad (21)$$

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

$$r_f^j \leq \frac{C \;\; l_f^j}{\sum_{n \in Q \wedge n \neq f} l_n^{max} + l_f^j - l_{max}} \quad (22)$$

To gain a qualitative understanding of (22), let $l_f^j = l_{max} = l_n^{max} = l$ and $r_f^j = r_f$. Then, $\Delta(p_f^j) \geq 0$ if $r_f \leq \frac{C}{|Q|-1}$. That is, maximum delay of packets of a flow in SFQ is smaller than in WFQ if the link bandwidth used by the flow is at most $\frac{C}{|Q|-1}$; such a flow is referred to as a low-throughput flow. This is also illustrated by Figure 1(a), which plots the reduction in delay in SFQ for different number of flows and flow rates, assuming 200 byte packets and link capacity of 100Mb/s. As the figure shows, whereas the delay reduces for flows with rate $r_f < \frac{C}{|Q|-1}$, i.e., low throughput flows, it increases for flows with rate $r_f \geq \frac{C}{|Q|-1}$, i.e., high throughput flows. To compare the delay performance of WFQ and SFQ in an example scenario, 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.

SFQ is also expected to lower the average delay of low-throughput applications while increasing the average delay of high-throughput ones. 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 1(b) 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 higher in WFQ than in SFQ; at 80.81% link utilization, the average delay is $4.7ms$ higher in WFQ than in SFQ.

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 desirable when sources may be malicious [4], independence of delay guarantee from traffic characterization enables a server to provide various QoS guarantees to flows conforming to any specification [10]. To enable a network of servers to provide similar guarantees, we derive end-to-end delay guarantee in Section 2.3.1.

2.2.2 Delay-cum-throughput Guarantee

We first establish a general property of SFQ FC and EBF servers in Theorems 4 and 5, respectively, and then derive their delay-cum-throughput guarantees in Corollaries 1 and 2.

Theorem 4 If the capacity of a SFQ FC server with parameters $(C, \delta(C))$ is not exceeded, then:

$$L_{SFQ}(p_f^j) \leq t + \sum_{n=k-1}^{n=j-1} \frac{l_f^n}{r_f^n} + \theta_f^j \quad (23)$$

where $t \geq A(p_f^j)$ and packet p_f^k is the first packet in the queue of flow f at time t.

Theorem 5 If the capacity of a SFQ EBF server with parameters $(C, B, \alpha, \delta(C))$ is not exceeded, then:

$$P\left(L_{SFQ}(p_f^j) \leq t + \sum_{n=k-1}^{n=j-1} \frac{l_f^n}{r_f^n} + \theta_f^j + \gamma\right) \geq 1 - Be^{-\alpha\gamma} \quad (24)$$

where $t \geq A(p_f^j)$ and packet p_f^k is the first packet in the queue of flow f at time t.

Corollaries 1 and 2 use Theorems 4 and 5, respectively, to derive the delay-cum-throughput guarantees of SFQ FC and EBF servers, respectively.

Corollary 1 If the capacity of a SFQ FC server with parameters $(C, \delta(C))$ is not exceeded, then:

$$L_{SFQ}(p_f^j) \leq \max\left\{L_{SFQ}(p_f^{j-1}), A(p_f^j)\right\} + \frac{l_f^{j-1}}{r_f^{j-1}} + \theta_f^j \quad (25)$$

where $L_{SFQ}(p_f^0) = 0$.

Corollary 2 If the capacity of a SFQ EBF server with parameters $(C, B, \alpha, \delta(C))$ is not exceeded, then

$$P\left(L_{SFQ}(p_f^j) \leq \max\left\{L_{SFQ}(p_f^{j-1}), A(p_f^j)\right\} + \frac{l_f^{j-1}}{r_f^{j-1}} + \theta_f^j + \gamma\right) \geq 1 - Be^{-\alpha\gamma} \quad (26)$$

where $L_{SFQ}(p_f^0) = 0$.

To observe the advantages of delay-cum-throughput guarantee over delay guarantee, consider a 10 Mb/s constant rate SFQ server that is serving 10 flows, each with packet size of 200 bytes and reserved rate 1 Mb/s (i.e., for all flows n, $r_n^j = r_n = 1$Mb/s). Let N flows (including flow f) be continuously backlogged and the rest of the flows

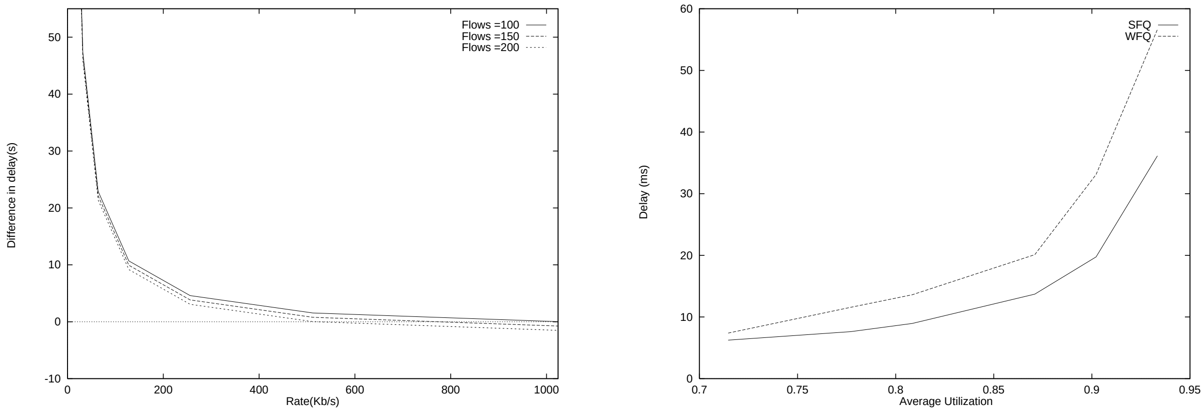

Figure 1: (a) Difference in maximum delay in WFQ and SFQ (b) Comparison of average delay in WFQ and SFQ

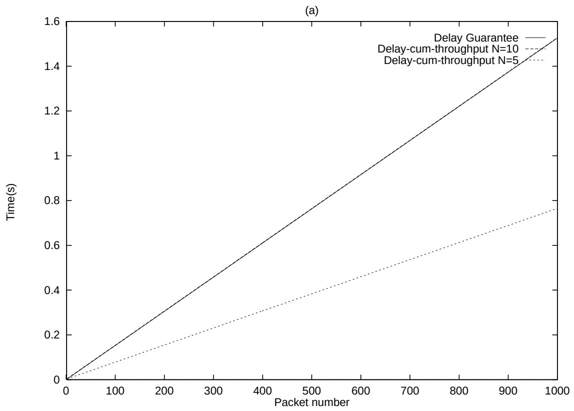

Figure 2: Bounds derived using delay and delay-cum-throughput guarantee for different number of backlogged flows.

send no packets. Since only N flows are backlogged and all flows have the same weight, flow f receives an effective throughput of $\frac{10}{N}$Mb/s. Hence, let departure time of p_f^j be $j\frac{200\ bytes}{\frac{10}{N}Mb/s}$. Figure 2 plots the bounds on departure time of packet p_f^{j+1} obtained using delay guarantee and delay-cum-throughput guarantee for $j \geq 1$ and $N = 10$ and $N = 5$. As the figure illustrates, when all the flows are backlogged, i.e., $N = 10$, the bound derived using delay guarantee is tighter. However, when only five flows are backlogged, i.e., $N = 5$, then the bound derived using delay-cum-throughput guarantee is significantly better. Hence, delay-cum-throughput guarantee improves upon the bounds of delay guarantee when the actual service received by a flow is better than the service that has been guaranteed.

In networks that carry traffic with multiple time-scale variation (for example, video traffic), many flows will receive service better than that guaranteed by the network. Hence, the improved bounds yielded by delay-cum-throughput guarantee are desirable. In Section 2.3 we derive the delay-cum throughput guarantee of a network of servers and illustrate the potential utility of the improved bounds yielded by delay-cum-throughput guarantee for flow controlled data and adaptive real-time applications.

2.3 End-to-End Deadline Guarantee

In this section, we utilize the single server deadline guarantee to derive delay and delay-cum-throughput guarantee of a network of servers.

2.3.1 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 [10]. 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 [10] to derive end-to-end delay guarantee of a network of servers that employ algorithms in the class of Guaranteed Rate (GR) scheduling algorithms (the framework presented in [7] can also be employed to study the end-to-end behaviour). However, the end-to-end delay guarantee presented in [10] 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 [10].

Observe that SFQ delay guarantee for both FC and EBF servers when the server capacity is not exceeded can be rewritten as:

$$P\left(L_{SFQ}(p_f^j) \leq EAT(p_f^j, r_f^j) + \beta_f^j + \gamma\right) \geq 1 - Be^{-\lambda\gamma} \quad (27)$$

Substituting $\beta_f^j = \theta_f^j$, $B = 0$ and $\lambda = \infty$, yields the delay guarantee for FC server. Substituting $\beta_f^j = \theta_f^j$ and $\lambda = \alpha$, yields the delay guarantee for EBF servers. Hence, we will use (27) 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 (27). 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 3.

Let τ^i 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:

$$P\left(L^i(p^j) \leq EAT^i(p^j, r^{j,i}) + \beta^{j,i} + \gamma\right) \geq 1 - B^i e^{-\lambda^i \gamma} \tag{28}$$

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

$$\begin{aligned} P \quad (\quad & EAT^{i+1}(p^j, \widehat{r}^{j,i}) \leq EAT^i(p^j, \widehat{r}^{j,i}) \\ & + \max_{n \in [1..j]} \{\beta^{n,i}\} + \tau^i + \gamma) \geq 1 - B^i e^{-\lambda^i \gamma} \end{aligned} \tag{29}$$

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

Corollary 3 If scheduling algorithm at each server on the path of a flow satisfies (28), and there are K servers on the path of the flow, then:

$$\begin{aligned} P\left(L^K(p^j)\right. & \leq EAT^1(p^j, \widehat{r}^j) + \sum_{n=1}^{n=K} \max_{m \in [1..j]} \{\beta^{m,n}\} \\ & + \Psi^{K-1} + \gamma) \geq 1 - \Phi^K e^{-\gamma \Lambda^K} \end{aligned} \tag{30}$$

where $L^K(p^j)$ is the time at which packet p^j leaves server K, $\widehat{r}^j = \min_{n \in [1..K]} \widehat{r}^{j,n}$, $\Psi^{K-1} = \sum_{n=1}^{n=K-1} \tau^n$, $\Phi^K = \sum_{n=1}^{n=K} B^n$, and $\Lambda^K = \frac{1}{\sum_{n=1}^{n=K} \frac{1}{\lambda^n}}$.

To derive Corollary 3, we have only required the scheduling algorithm at each server to satisfy (28). Hence, any scheduling algorithm that satisfies (28) (for example, Virtual Clock, WFQ, and SCFQ) can interoperate to provide end-to-end guarantee. Furthermore, Corollary 3 can be used for an internetwork of FC and EBF servers. Finally, the proof method of Theorem 6 and Corollary 3 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.

2.3.2 End-to-End Delay-cum-throughput Guarantee

When a flow is served by a network of servers, a destination knows the departure time of a packet from the last server. Furthermore, from the traffic characteristics of a flow, it may also know the arrival time of a packet at the first server on the path. Hence, the objective is to determine a bound on the departure time of a packet from the last server based on its arrival time at the first server and departure time of the previous packet at the last server.

Observe that SFQ delay-cum-throughput guarantee for both FC and EBF servers when the server capacity is not exceeded can be rewritten as:

$$\begin{aligned} P\left(L_{SFQ}(p_f^{j+1}) \leq \right. & \max\{L_{SFQ}(p_f^j, A(p_f^{j+1})\} \quad + \beta_f^{j+1} \\ & + \gamma) \geq 1 - B e^{-\lambda \gamma} \end{aligned} \tag{31}$$

Substituting $\beta_f^{j+1} = \frac{l_f^j}{r_f^j} + \theta_f^{j+1}$, $B = 0$ and $\lambda = \infty$, yields the delay-cum-throughput guarantee for FC server. Substituting $\beta_f^{j+1} = \frac{l_f^j}{r_f^j} + \theta_f^{j+1}$ and $\lambda = \alpha$ yields the delay-cum-throughput guarantee for EBF servers. Hence, we will use (31) to derive the end-to-end delay-cum-throughput 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 (27).

Let $\hat{\tau}^i$ denote the lower bound on the propagation delay between servers i and $i+1$. As in the previous section, we drop the subscript f from all the variables and identify all variables of i^{th} server by superscript i. Theorem 7 establishes the end-to-end delay-cum-throughput guarantee.

Theorem 7 If there are K servers on the path of a flow, and each server i guarantees that:

$$\begin{aligned} P\left(L^i(p^{j+1})\right. & \leq \max\{L^i(p^j), A^i(p^{j+1})\} + \beta^{j+1,i} + \gamma) \\ & \geq 1 - B^i e^{-\lambda^i \gamma} \end{aligned} \tag{32}$$

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

$$\begin{aligned} P(L^i(p^{j+1}) \leq \quad & \max\{L^K(p^j) - \widehat{\Psi}^{K-1} \quad , A^1(p^{j+1})\} + \\ \Psi^{i-1} \quad & + \sum_{n=1}^{n=i} \beta^{j+1,i} + \gamma) \geq \quad 1 - \Phi^i e^{-\gamma \Lambda^i} \end{aligned} \tag{33}$$

where $\widehat{\Psi}^{K-1} = \sum_{n=1}^{n=K-1} \hat{\tau}^k$, $\Psi^{i-1} = \sum_{n=1}^{n=i-1} \tau^k$, $\Phi^i = \sum_{n=1}^{n=i} B^n$, and $\Lambda^i = \frac{1}{\sum_{n=1}^{n=i} \frac{1}{\lambda^n}}$.

If all the servers are FC servers and provide deterministic guarantee, then (33) simplifies to:

$$\begin{aligned} L^K(p^{j+1}) \leq \quad & \max\{L^K(p^j) - \widehat{\Psi}^{K-1}, A^1(p^{j+1})\} + \\ & \sum_{n=1}^{n=K} \beta^{j+1,i} + \Psi^{K-1} \end{aligned} \tag{34}$$

for $i = K$. If a destination knows the relationship between the arrival time of packet p^{j+1} at the first server and departure time of p^j at the last server (possibly from the traffic characteristics of the source) and the service received by a flow is better than that guaranteed by the network, then just like in the case of a single server, the destination can utilize (34) to derive bounds on packet delay better than those determined by end-to-end delay guarantee. In particular, if for all packets p^{j+1}, $L^K(p^j) =$

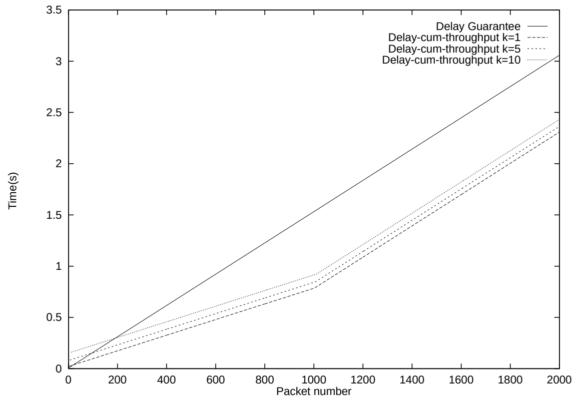

Figure 3: Bounds derived using delay and delay-cum-throughput guarantee for different values of k.

$\max\{L^K(p^j) - \sum_{n=1}^{n=K-1} \hat{\tau}^k, A^1(p^{j+1})\}$, then by recursive use of (34), we get:

$$L^K(p^{j+k}) \leq L^K(p^j) + \sum_{m=1}^{m=k} \left(\sum_{n=1}^{n=K} \beta^{j+m,i} + \Psi^{K-1} \right) \tag{35}$$

To observe the advantage of bounds derived using (35), consider a flow that is served by 5 servers. Let each server be a constant rate server with rate $C = 10Mb/s$ and for ease of exposition, let there be zero propagation delay between them. Let each server serve $N = 10$ flows, each with reserved rate 1Mb/s and packet size of 200 bytes. Also, let the flow be continuously backlogged at the first server. Let $L^K(p^j)$ be given as:

$$L^K(p^j) = \sum_{n=1}^{n=j-1} \frac{l}{r^j} + K\alpha \tag{36}$$

where $l = 200$ bytes, $K = 5$ and $\alpha = \frac{Nl}{C}$. Let $r^j = 2Mb/s$ for the first 1000 packets, i.e., let them receive service better than that guaranteed by the network, and let $r^j = 1Mb/s$ for $j > 1000$. Figure 3 plots the bound on the departure time of packet p^{j+k} ($k = 1, 5, 10$) for different values of j using (35) as well as the end-to-end delay guarantee for this scenario. As the figure demonstrates, (35) improves upon the bounds of delay guarantee and tracks the actual arrival time of packets much more closely.

We envision the end-to-end delay-cum-throughput guarantee to be useful for at least two classes of applications:

- Flow controlled data applications: Consider a flow controlled data source that reserves a minimum rate at each of the servers on the path to the destination. To increase its throughput by taking advantage of statistical multiplexing of various sources, let the source estimate the bottleneck rate, which is at least the reserved rate, and send at the estimated bottleneck rate [15]. Due to the fluctuations in the bottleneck rate as well as the inherent delay and errors in the estimation process, such a source may send at a rate higher than the bottleneck rate. This will lead to queue build up at the bottleneck server and eventually packet losses. Let packets $p^j, ..., p^{j+k}$ be lost due to buffer overflow. In the simplest case, a destination can detect loss of these packets only on arrival of packet p^{j+k+1}. However, if the network provides delay-cum-throughput guarantee, then the destination can use (35) to determine a bound on the arrival time of packets $p^j, ..., p^{j+k}$ and declare them lost if they do not arrive by then. It can thus detect packet losses earlier than the arrival of packet p^{j+k+1}. The early detection of packet losses can be used by a destination to "close" the feedback loop between a source and destination faster and thus improve the throughput of the source.
- Adaptive, real-time, playback applications: Applications such as audio and video that can tolerate discontinuities in playback and adapt their playback point as per the network congestion, may reserve a minimum rate and send packets at a higher rate. In such a scenario, a destination can use the delay-cum-throughput guarantee to determine a bound on the arrival time of future packets and use the bounds to suitably adapt the playback point.

The algorithms and protocols that exploit the advantages of delay-cum-throughput guarantee for these and other applications is the subject of ongoing research and beyond the scope of this paper.

2.4 Discussion

SFQ borrows the concept of "self-clocking" and scheduling packets in the increasing order of start tags from SCFQ and FQS, respectively. However, it leads to better performance than either of the two. SFQ has the same fairness measure and implementation complexity as SCFQ but has smaller delay guarantee. Similarly, whereas FQS is unfair over variable rate servers and has high implementation complexity, SFQ is fair over variable rate servers and has lower implementation complexity. Furthermore, in FQS, since 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 is at least that in SFQ.

The delay guarantee of SFQ depends on the maximum packet length of all the flows at the server. In contrast, the delay guarantee of WFQ depends only on the flow's properties. Thus, WFQ provides better isolation of delay guarantee of a flow. We have shown in [12] that the delay guarantee of SFQ is similar to that of an online algorithm[4] that minimizes unfairness. Furthermore, as we demonstrated in Section 2.2.1, it is the lack of isolation of delay guarantee that enables SFQ to provide lower delay to low throughput flows at the expense of increased delay to high throughput flows. However, if SFQ is employed to provide a priori specified bounds on packet delay, then the maximum number of flows as well as their packet sizes would have to be estimated. In some networking environments, such an esti-

[4] An online scheduler is one which does not use the length of packet p_f^{k+1} in making a scheduling decision for packet p_f^k.

mate may be large and consequently SFQ may not be able to provide lower a priori delay to low throughput applications. In such a case, low delay to low throughput flows may be provided by employing:

- Fair scheduling algorithms that allocate only rate and have delay guarantee similar to WFQ: In such a case, low delay is provided to low throughput flows by reserving higher rate. This may result into low utilization of the network. However, the main advantage of such algorithms is that they have $O(1)$ complexity admission control algorithms.
- Fair scheduling algorithms that achieve separation of rate and delay allocation: In such a case, the network utilization is higher. However, these algorithms have $O(Q)$ complexity admission control algorithms [7].

For networking environments where either of these two approaches are preferable over SFQ, we have designed a class of Fair Airport (FA) algorithms [12]. An algorithm in FA class combines SFQ with any non-work conserving algorithm in Rate Controlled Service Discipline (RCSD) class [7]. By appropriately choosing an algorithm from RCSD class, fair algorithms that either allocate only rate or achieve separation of rate and delay allocation can be designed. This method leads to the design of the *first* fair algorithm that achieves separation of rate and delay allocation. The property of SFQ that it does not use the length of a packet in determining its priority is central to the design of such FA algorithms. Though FA algorithms have higher implementation complexity than SFQ, they can be efficiently implemented. Furthermore, they are fair over FC servers. The detailed presentation of FA algorithms is beyond the scope of this paper.

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, on an average, is within 11% of the lower bound; 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, 18]. 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 services. 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. Hierarchical link sharing can also be employed to support a *link-sharing* service in which the bandwidth of a link is partitioned among several organizations and the bandwidth of an organization is recursively partitioned among its suborganizations [18].

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 benefits 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; and best effort services may use a fair scheduler for throughput intensive flow-controlled data applications.

The requirements of hierarchical link sharing are 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 [18]. 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.

Since SFQ allocates bandwidth fairly even over variable rate servers, it can be employed for achieving hierarchical link sharing. In what follows, we present a hierarchical SFQ scheduler.

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. If the leaf class is an aggregation of flows, it schedules flows by employing a leaf class dependent scheduler (see [9] for an implementation of hierarchical SFQ scheduler). 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 (a similar hierarchical WF^{2}Q+ scheduler has been independently presented in [1]). Moreover, in contrast to link sharing mechanism in [6], it provides bounds on various performance measures. To derive bounds on the performance measures, we first prove the following corollaries of Theorems 4 and 5. Let $T_f(w)$ denote the time taken to serve flow f packets of aggregate length w which are served in the same backlogged period of the flow.

Corollary 4 If the capacity of a SFQ FC server with parameters $(C, \delta(C))$ is not exceeded and $r_f^j = r_f$ for all packets, then $T_f(w)$ is given as:

$$T_f(w) \leq \frac{w}{r_f} + \frac{l_f^{max}}{r_f} - \alpha_f + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \delta(C) \quad (37)$$

where $\alpha_f = \min\{\frac{l_f^j}{r_f} - \frac{l_f^j}{C}\}$ and the minimum is over all the flow f packets.

Corollary 5 If the capacity of a SFQ EBF server with parameters $(C, B, \alpha, \delta(C))$ is not exceeded and $r_f^j = r_f$ for all packets, then random variable $T_f(w)$ is given as:

$$P\left(T_f(w) \leq \frac{w}{r_f} + \frac{l_f^{max}}{r_f} - \alpha_f + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \delta(C) + \gamma\right) \geq 1 - Be^{-\alpha\gamma} \quad (38)$$

where $\alpha_f = \min\{\frac{l_f^j}{r_f} - \frac{l_f^j}{C}\}$ and the minimum is over all the flow f packets.

Now, 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 Corollary 4 we conclude that the virtual server corresponding to f is a FC server with parameters:

$$\left(r_f, \frac{l_f^{max}}{r_f} - \alpha_f + \frac{\sum_{n \in Q \wedge n \neq f} l_n^{max}}{C} + \delta(C)\right) \quad (39)$$

Similarly, using Corollary 5, we conclude that if the link is an EBF server, then the virtual server corresponding to f is a EBF server. 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, the bounds on deadline and end-to-end deadline guarantee of a flow when it is hierarchically scheduled can be determined as follows:

- **Deadline Guarantee**: Since each of the virtual server is either FC or EBF server, Theorems 2 and 3 can be used to determine the single server delay guarantee and Corollaries 1 and 2 can be used to determine the single server delay-cum-throughput guarantee of the flows.
- **End-to-End Deadline Guarantee**: Since the single server deadline guarantee when a flow is hierarchically scheduled satisfies (28) and (32), Corollary 3 and Theorem 7 can be used to determine the end-to-end deadline guarantee.

The above analysis method is general and can be employed for any fair scheduling algorithm that provides guarantees similar to SFQ, i.e., bounds on $T(w)$ over FC and EBF servers. Furthermore, this analysis is tighter than the analysis presented in [1, 13]. To observe this, consider a tree with three classes; two leaf classes and a root class. Let the rate of leaf classes 1 and 2 be r_1 and r_2, respectively, and let each of them contain 2 flows with equal weights. Let both the leaf classes be scheduled by SFQ and the length of all packets be l. Then, it can be shown that the best bound on delay of packet p_f^j for flow f in leaf class 1 using the analysis in [1, 13] is:

$$EAT(p_f^j, \frac{r_1}{2}) + (\frac{l}{r_1} + \frac{2l}{C}) + \frac{2l}{r_1} \quad (40)$$

In contrast using our analysis, we get:

$$EAT(p_f^j, \frac{r_1}{2}) + \frac{2l}{C} + \frac{2l}{r_1} \quad (41)$$

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 8, 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. Delay EDD on arrival of packet p_f^j of flow f assigns it a deadline, denoted by $D(p_f^j)$, and schedules packets in increasing order of deadline [5]. $D(p_f^j)$ is defined as:

$$D(p_f^j) = EAT(p_f^j, r_f) + d_f \quad (42)$$

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

Theorem 8 If Q is the set of flows serviced by the server and

$$\forall t > 0 : \quad \sum_{n \in Q} \max\{0, \left\lceil \frac{(t - d_n) r_n}{l_n} \right\rceil \frac{l_n}{C}\} \leq t \quad (43)$$

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:

$$L_{EDD}(p_f^j) \leq D(p_f^j) + \frac{l_{max}}{C} + \delta(C) \quad (44)$$

Due to high computational complexity, it may not be feasible to employ (43) as the schedulability test. Hence, conditions stronger than (43) which have lower computational complexity have been developed in [22]. The theorem holds under the stronger conditions as well.

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 4). The driver is used to maintain weights for connections. 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 as its parameters.

To experimentally validate the implementation of the scheduler, we initiated three connections with weights 1, 2, and 3. Each of the connections terminated after transmitting 500,000 4KB packets. Figure 4 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. Observe from Figure 4 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 fairness guarantee that is close to the best achievable guarantee. We analyzed its single server and end-to-end deadline 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. 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 and demonstrated that our analysis leads to tighter results.

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.

Acknowledgements

We would like to thank the anonymous reviewers for their constructive comments that have improved the presentation of the paper. This research was supported in part by IBM Graduate Fellowship, IBM Faculty Development Award, Intel, the National Science Foundation (Research Initiation Award CCR-9409666 and CAREER award CCR-9624757), NASA, Mitsubishi Electric Research Laboratories (MERL), and Sun Microsystems Inc.

References

[1] J.C.R. Bennett and H. Zhang. Hierarchical Packet Fair Queuing Algorithms. In *Proceedings of SIGCOMM'96*, pages 143–156, August 1996.

[2] J.C.R. Bennett and H. Zhang. WF^2Q: Worst-case Fair Weighted Fair Queuing. In *Proceedings of INFOCOM'96*, pages 120–127, March 1996.

[3] J. Davin and A. Heybey. A Simulation Study of Fair Queueing and Policy Enforcement. *Computer Communication Review*, 20(5):23–29, October 1990.

[4] 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.

[5] D. Ferrari and D. C. 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.

[6] S. Floyd and V. Jacobson. Link-sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995.

[7] L. Georgiadis, R. Guerin, V. Peris, and K.N. Sivarajan. Efficient Network QoS Provisioning Based on per Node Traffic Shaping. In *Proceedings of INFOCOM'96*, pages 102–110, March 1996.

[8] S.J. Golestani. A Self-Clocked Fair Queueing Scheme for High Speed Applications. In *Proceedings of INFOCOM'94*, pages 636–646, April 1994.

[9] P. Goyal, X. Guo, and H.M. Vin. A Hierarchical CPU Scheduler for Multimedia Operating Systems. In *Proceedings of Operating System Design and Implementation (OSDI'96), Seattle*, pages 107–122, October 1996.

[10] P. Goyal and H. M. Vin. Generalized Guaranteed Rate Scheduling Algorithms: A Framework. In *IEEE/ACM Transactions on Networking (to appear)*. Also available as technical report TR95-30, Department of Computer Sciences, The University of Texas at Austin.

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

[12] 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, Department of Computer Sciences, The University of Texas at Austin, January 1996. Available via URL http://www.cs.utexas.edu/users/dmcl.

[13] P. Goyal, H. M. Vin, and H. Cheng. Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks. In *Proceedings of ACM SIGCOMM'96*, pages 157–168, August 1996.

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

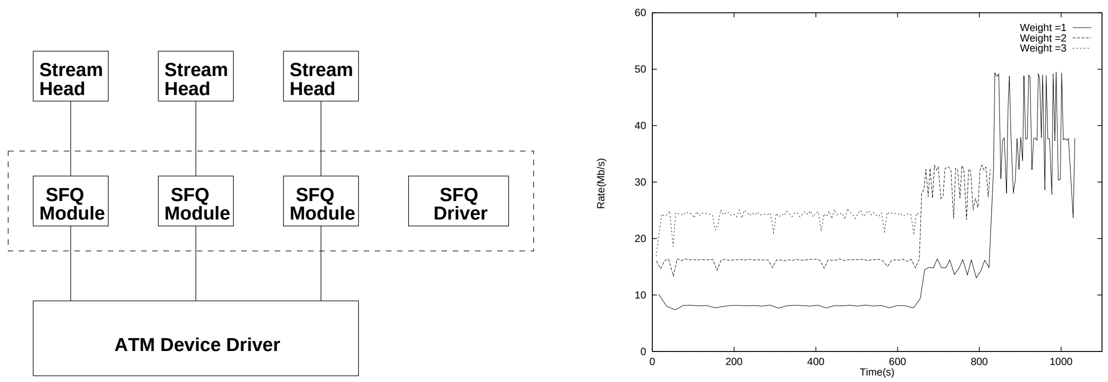

Figure 4: (a) SFQ scheduler implementation (b) Throughput of the connections

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

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

[17] 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.

[18] 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.

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

[20] D. Stiliadis. *Traffic Scheduling in Packet-Switched Networks: Analysis, Design and Implementation*. PhD thesis, Department of Computer Science and Engineering, University of California at Santa Cruz, 1996.

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

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