Real-Time Block Transfer Under a Link Sharing Hierarchy*

Geoffrey G. Xie
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943
xie@cs.nps.navy.mil

Simon S. Lam
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188
lam@cs.utexas.edu

Abstract

Most application-level data units are too large to be carried in a single packet (or cell) and must be segmented for network delivery. To an application, the end-to-end delays and loss rate of its data units are much more relevant performance measures than ones specified for individual packets (or cells). The concept of a burst (or block) was introduced to represent a sequence of packets (or cells) that carry an application data unit. In this paper, we describe how a real-time VBR service, with QoS parameters for block transfer delay and block loss rate, can be provided by integrating concepts and delay guarantee results from our previous work on burst scheduling, together with ideas from ATM block transfer. Two new contributions are presented herein. First, we design an admission control algorithm to provide the following classes of service: bounded-delay block transfer with no loss, and bounded-delay block transfer at a specified block loss rate. Second, we show how to extend existing end-to-end delay bounds to networks with hierarchical link sharing.

1 Introduction

To an application, the end-to-end delays of its data units are much more relevant performance measures than ones specified for individual packets or cells. For example, a video picture (or file) being sent by an application over an IP network may be segmented into a sequence of IP datagrams. The delay incurred to deliver the entire video picture (or file) is much more important to the application than the delays of individual IP datagrams. As another example, an IP datagram carrying an email message may be segmented into a sequence of cells for delivery over an ATM network. The delay incurred to deliver the entire email message is more important than the delays of individual cells. From this observation, we introduced the concept of a burst to represent a sequence of packets that carry an application level data unit, and designed the class of burst scheduling networks to provide delay guarantees to bursts [8, 9].

The *ATM block transfer* (ABT) capability being standardized by ITU-T is based upon a similar observation [7]. The objective of ABT is to allow a data source to dynamically negotiate its bandwidth reservation on the basis of a block of cells. Note that a higher-layer protocol data unit, fragmented into a number of ATM cells, is lost if any one of its cells is lost. Therefore, even a low cell loss rate can cause a significant loss rate for the higher-layer protocol. As a result, the higher-layer protocol's throughput may be much less than the protocol session's throughput measured in delivered cells. The concept of a block was introduced to represent a sequence of cells, which may contain a single data unit or multiple data units for the higher-layer protocol. A block is bracketed by two RM cells. A leading RM cell requests a reserved bandwidth for the block, and a trailing RM cell releases the reserved bandwidth. Cells are handled in blocks by a switch. In particular, a block of cells is either discarded or accepted entirely.

For the ABT service, the concept of cell loss rate can be generalized to *block loss rate*. Such a generalization is backward-compatible with the existing ATM TM 4.0 service architecture [5] since a block is a sequence of cells, with a single cell being a special case. Similarly, the concept of cell transfer delay for real-time VBR services can be generalized to *block transfer delay* which, we believe, is a more relevant performance measure to many applications; for example, if every picture in a video sequence is carried by a block of cells, then the block transfer delays are the same as picture delays.

In this paper, we describe how a real-time VBR service called *real-time block transfer* with QoS parameters for block transfer delay and block loss rate can be provided by integrating concepts and delay guarantee results from our previous work on burst scheduling [8]

*Most of this work was done while G. Xie was a graduate student in the Department of Computer Sciences, the University of Texas at Austin. Research supported in part by National Science Foundation grant no. NCR-9506048, an Intel Graduate Fellowship, and the Texas Advanced Research Program grant no. 003658-220.

and group priority [9], together with ideas from ATM block transfer. In particular, we have designed an admission control algorithm for real-time block transfer, which provides the following service classes: bounded-delay block transfer with no loss (deterministic delay guarantee), and bounded-delay block transfer at a specified block loss rate.

It is envisioned that future integrated services networks will support not only link sharing by multiple service classes (real-time service, best-effort service, etc.) but also by multiple administrative classes (different agencies and organizations) [4]. Specifically, packets (or cells) from sessions belonging to different service classes and administrative classes interact with one another when they are statistically multiplexed at an output link of a switch. The switch's packet scheduling algorithm plays an important role in controlling such link sharing. Hierarchical link sharing has been proposed as a solution [1]. In this paper, we also describe a general approach for extending the end-to-end delay bounds for bursts[1] from networks in which links are shared by service classes only [8, 9] to networks in which links are hierarchically shared (e.g., by administrative classes first, and then by service classes within each administrative class). Specifically, we have proved two theorems that can be used to derive block delay guarantees by a *fluctuation constrained* server from block delay guarantees by a constant-rate server. The theorems are general. They are proved for a large class of well known servers.

The balance of this paper is organized as follows. In Section 2, we describe the basic idea of block-based admission control for ABT and the concept of burst scheduling. The end-to-end delay bounds of burst scheduling are shown. In Section 3, we design a flow level admission control algorithm for real-time block transfer services. We evaluated the algorithm using a simulator driven by MPEG video traces, and some experimental results are presented in this section. In Section 4, we introduce hierarchical link sharing. We model each logical server in a link sharing hierarchy as a fluctuation constrained (FC) server, and show how to extend end-to-end delay bounds such as the ones presented in Section 2 to networks with hierarchical link sharing.

2 Real-time Block Transfer

2.1 Block-based Admission Control

In ABT, dynamic bandwidth reservation and allocation for a block of cells can be carried out in two ways: (i) ABT with Delayed Transmission (ABT/DT), and (ii) ABT with Immediate Transmission (ABT/IT). For our discussion, we focus on the latter. In ABT/IT, the block is sent *immediately* after a preceding RM cell, which contains a request for a cell rate. The block proceeds on a switch by switch basis, with each switch either forwarding the block with guaranteed QoS for *every cell* in the block or discarding the *entire* block if a required resource such as bandwidth is not available. In other words, the switches perform admission control on a block by block basis.

With block-based admission control, cell losses are concentrated over a small number of blocks, and bandwidth is not wasted on delivery of partial blocks. (This is similar to the idea of early packet discard in recent studies on IP over ATM [13].) Therefore, ABT is able to avoid the situation in which cell losses spread over a large number of higher-layer data units causing throughput degradation of such data units.

With block-based admission control, ABT is also able to offer QoS measured in terms of blocks. In particular, the concept of cell loss rate can be generalized to *block loss rate*.

2.2 Burst Scheduling

For burst scheduling networks, we model a flow as a sequence of bursts [8]. A burst, a sequence of packets, corresponds to a block in ABT with some minor differences in detail. In particular, instead of two special packets being used, the first packet of each burst is marked and stored in it information on the size of the burst (in number of packets) and the average rate of the burst. Moreover for efficient packet scheduling and delay jitter control, packets of each burst satisfy a jitter timing constraint [8].

The following delay bound results are taken from [8] and [9].

Consider a flow that traverses a sequence of nodes, indexed by $0, 1, 2, \ldots, K+1$, where node 0 denotes the source, node $K+1$ the destination, and the other nodes packet switches. If the capacity of every link on the network path is not exceeded, then the end-to-end delay[2] of every burst m of the flow, denoted by D_m, $m = 1, 2, \ldots$, has the following upper and lower bounds:

$$D_m \leq \frac{b_m + g_m}{\lambda_m} + (K-1)\max_{1 \leq n \leq m}\{\frac{g_n}{\lambda_n}\} + \sum_{k=1}^{K}\alpha_k \quad (1)$$

$$D_m \geq (K-1)\frac{g_m}{\lambda_m} + \sum_{k=1}^{K}\alpha_k, \quad (2)$$

where we use the following notation

[1] End-to-end delay bounds for blocks are obtained by specifying a burst [8, 9] to represent a block of ATM cells.

[2] measured from the time when the first cell enters the network to the time when the last cell leaves the network.

b_m	*size of burst m (packets)*
λ_m	*average rate of burst m (packets/second)*
g_m	*group size (in packets) chosen for burst m by the switches for efficient scheduling;* $1 \leq g_m \leq b_m$
α_k	*a small constant associated with the link from switch k to k + 1*

Our design of burst scheduling networks consists of specialized components such as source and flow regulators. But the concept of burst scheduling, i.e. modeling a flow as a sequence of bursts and providing bounded delays to bursts (blocks) when link capacity is not exceeded, is quite general. It can be realized by many designs other than ours. In what follows, we refer to burst scheduling as a general concept independent of the design details.

2.3 Integration of ABT and Burst Scheduling

ABT is able to minimize block losses through the use of block-based admission control, and provide block loss rate as QoS. Burst scheduling networks provide bounded block transfer delays as QoS when link capacity is not exceeded. Integrating the concepts and results from ABT and burst scheduling, we define a real-time VBR service, called real-time block transfer, that provides the following two classes of services: (i) bounded-delay block transfer with zero block loss, and (ii) bounded-delay block transfer at a specified block loss rate.

Admission control at the flow level is the key for our integration. For class (i), peak rate reservation can be used for admission control to ensure that the link capacity allocated to this class is not exceeded without discarding blocks. For class (ii), overbooking of the class's allocated capacity is allowed at the time of connection setup while block-based admission control is used to ensure that link capacity is not exceeded at any time by discarding blocks if necessary. To limit the block loss rate to a specified value, the extent of overbooking is controlled by flow level admission control. In the next section, we describe in detail a flow level admission control algorithm for the real-time block transfer services.

3 Admission Control

In this section, we design an algorithm for flow level admission control at connection setup time for real-time block transfer.[3] Consider a flow whose source requests for a real-time block transfer service. We assume that at the time of connection setup, the source supplies the network two sets of flow parameters: (i) QoS parameters: block loss probability BLP and block transfer delay bound BTD,[4] and (ii) traffic parameters: sustained cell rate SCR, peak cell rate PCR, and cell rate variation CRV. Whether or not to admit the flow is a decision made by each switch in the path of the flow. Specifically, each switch in the path accepts the flow only if doing so will not cause violation of QoS guarantees to accepted flows; the network admits the flow only if all switches in the path accept the flow.

3.1 System Model

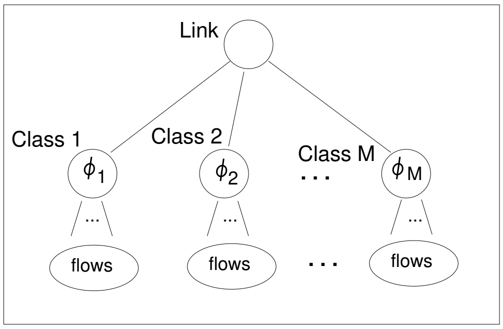

Figure 1: System model for admission control.

Our system model for admission control by a particular switch is shown in Figure 1. There are M classes of real-time block transfer service. They share a link[5] with capacity C bits/second. Each class is associated with a weight, ϕ_s for class s, which is a relative measure of its share of the bandwidth of the link. For ease of presentation and without loss of generality, we assume that $\sum_{s=1}^{M} \phi_s = 1$. Therefore, class s has a share of the link equal to $r_s = \phi_s \cdot C$ (bits/second). Each class offers a block loss rate, p_s for class s. Without loss of generality, we assume that

$$0 = p_1 < p_2 < \ldots < p_M < 1.$$

In other words, class 1 provides deterministic service, i.e., class (i) service defined in Section 2.3, and the other classes provide different levels of statistical service, i.e., class (ii) service defined in Section 2.3.

We also assume that an appropriate scheduling algorithm (such as WF^{2}Q+ [1]) is used by the link to provide a firewall between the service classes and guarantee each class its link share. (See Section 4.) As a result, admission control for each class can be performed independently.

3.2 Statistical Multiplexing

For flows requesting for the deterministic service, admission control should be based on peak rate reserva-

[3] We present the design in the context of ATM networks. Our design should be applicable to other types of networks.

[4] BTD will not be considered further in our design of admission control, assuming that an appropriate scheduling algorithm is used by the network to ensure burst delay guarantees.

[5] The link could be a logical one; see Section 4.

tion, which is necessary to ensure that the class's link share cannot be exceeded without discarding blocks.[6]

We next consider a statistical class s. Flows that request for this class of service can tolerate some block losses. Therefore their admission control can be more aggressive to increase utilization of the class's bandwidth by taking advantage of statistical multiplexing. In what follows, we derive a set of conditions that are sufficient to limit the block loss probability of the class at approximately p_s.

Assume that the service of the class is currently shared by a set of N flows (indexed by $1, 2, ..., N$). Let BLP_i, SCR_i, PCR_i and CRV_i be the QoS and traffic parameters[7] that are supplied to the switch by flow i. At any time, because of burst scheduling, at most one block from each flow is active [8]. Denote $\lambda_i(t)$ the reserved cell rate for the block of flow i that is active at time t. ($\lambda_i(t) = 0$ if there is no active block for flow i at t.) In our analysis below, $\lambda_i(t)$'s are considered independent random variables. Define

$$Y_s(t) = \sum_{i=1}^{N} \lambda_i(t) - r_s. \tag{3}$$

Consider a block that becomes active at time t. The switch immediately performs admission control on the block, and will discard it if $(Y_s(t) > 0)$. Therefore $\Pr(Y_s(t) > 0)$ is a good approximation of the block loss probability of the class. The goal of our analysis is to find conditions sufficient for

$$\Pr(Y_s(t) > 0) \leq p_s. \tag{4}$$

From a generalized version of the central limit theorem [11], we have

$$\frac{Y_s(t) - E[Y_s(t)]}{\sqrt{Var[Y_s(t)]}} \rightarrow \mathrm{N(0,1)}, \tag{5}$$

where N(0,1) is the standard normal distribution. Therefore, we can approximate $\Pr(Y_s(t) > 0) \leq p_s$ by

$$\Pr(X > \frac{-E[Y_s(t)]}{\sqrt{Var[Y_s(t)]}}) \leq p_s, \tag{6}$$

where $X \sim \mathrm{N(0,1)}$.
The following condition is sufficient for (6):

$$\frac{-E[Y_s(t)]}{\sqrt{Var[Y_s(t)]}} \geq Z_s, \tag{7}$$

where Z_s is the constant that satisfies $(\Pr(X > Z_s) = p_s)$. We have

$$E[Y_s(t)] = \sum_{i=1}^{N} E[\lambda_i(t)] - r_s \tag{8}$$

$$Var[Y_s(t)] = \sum_{i=1}^{N} Var[\lambda_i(t)]. \tag{9}$$

Define

$$Z = \frac{r_s - \sum_{i=1}^{N} E[\lambda_i(t)]}{\sqrt{\sum_{i=1}^{N} Var[\lambda_i(t)]}}. \tag{10}$$

Combining (7), (8), (9), and (10), we have the following approximate sufficient condition for (4):

$$\frac{Z_s}{Z} \leq 1. \tag{11}$$

We refer to the value of (Z_s/Z) as the Statistical Multiplexing Intensity (SMI) of class s. It should never exceed the threshold of 1 to limit the block loss rate at p_s. In practice, it is difficult to obtain the exact value of Z. However, Z can be estimated as follows:

$$\hat{Z} = \frac{r_s - \sum_{i=1}^{N} SCR_i}{\sqrt{\sum_{i=1}^{N} CRV_i}} \tag{12}$$

Note that the source of flow i may not have a good estimate of CRV_i at the time of connection setup. In such a case, CRV_i is upper bounded by $SCR_i \cdot (PCR_i - SCR_i)$, which can be used as a pessimistic estimate.

3.3 Algorithm Specification

The following admission control algorithm follows from the analysis in the previous section. The variables B_s and V_s are used to store respectively the available bandwidth and the total cell rate variance of class s. Initially $B_s = r_s$ and $V_s = 0$. We assume that if a source does not have a good estimate of CRV at the time of connection setup, it will let the network know by setting $CRV = 0$.

```
Admission_Control (BLP, PCR, SCR, CRV)
s := find_class(BLP);
if (s = 1 )
  then if (B1 - PCR >= 0)
         then accept the flow;
              B1 := B1 - PCR;
         else reject the flow;
  else if (CRV = 0)
         then CRV := SCR * (PCR - SCR);
       SMI := Zs * (sqrt(Vs + CRV)/(Bs - SCR));
       if (SMI <= 1)
```

[6] If the switch has a priori knowledge of the traffic characteristics of flows in the class, peak rate reservation may not be necessary. But it is unrealistic with today's networks.

[7] Note that BLP_i is the probability of block losses through one switch (i.e. a single hop) required for flow i. We will discuss how to distribute an end-to-end burst loss requirement to individual switches in a future report.

then accept the flow;
$B_s := B_s - SCR$;
$V_s := V_s + CRV$;
else reject the flow;

The algorithm is simple and straightforward. First, BLP is used to find the service class desired by the source. Assume the class found is s. If s is 1, the source requests for deterministic service. Therefore, the admission decision is based upon PCR of the flow and the bandwidth currently available for class 1. Otherwise, the source requests for a statistical service. The admission decision is then based on Z_s for class s, which has been calculated from p_s.

3.4 Experimental Results

We have performed a set of experiments to evaluate the performance of the admission control algorithm. We present the results in this section.

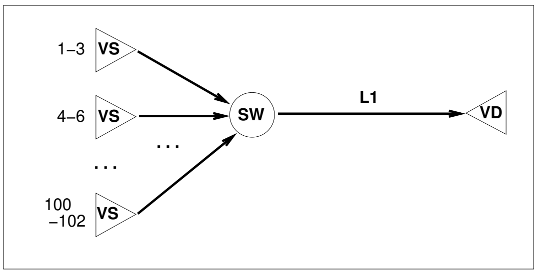

Figure 2: Simulated network.

The simulated network is shown in Figure 2. The nodes labeled by VS are video sources, and VD video destination. Each video source generates packets from a trace file obtained from one of the MPEG video sequences that we have downloaded from the Web.

Each video source makes reservation with the network before sending out data packets. The admission control algorithm is implemented for channel L1, and all video sources request for the same class of service with a targeted block loss rate, p. To evaluate the admission control algorithm, we varied the link bandwidth of L1 as well as the value of p. We ran each experiment for 10 seconds of simulated time.

In Figure 3, we show the channel utilization as a function of the targeted loss rate. Compared with deterministic guarantees, the network channel is used more efficiently for statistical guarantees; the price to pay is a small non-zero block loss probability. The utilization gain is more significant with a higher channel capacity, from below 30% to above 70% in the case where the capacity of L1 is 56 Mbps. This is because the improvement is due to statistical multiplexing gain, which is larger with a higher channel capacity.

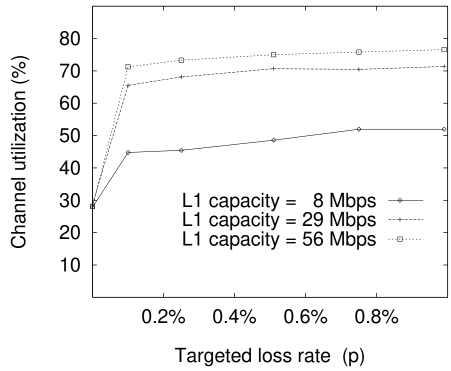

Figure 3: Channel utilization improvement

We also measured the actual block (picture) loss rate, averaged over five simulation runs using different random seeds. In Figure 4, a comparison of the actual and targeted loss rates is illustrated. The solid 45 degree line represents a perfect prediction by the central limit theorem. From the figure, we conclude that our admission control algorithm predicts the actual loss rate well when a large number of flows share the channel. (Around 30 flows were admitted when the channel capacity of L1 is 56 Mbps.) This agrees with our analysis; the larger the number of flows sharing the channel, the better the approximation based upon the central limit theorem.

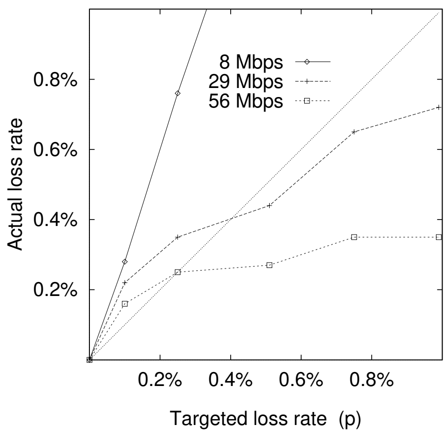

Figure 4: Actual vs. targeted loss rate

4 Hierarchical Link Sharing

Existing delay bounds for blocks (bursts), e.g. the ones presented in Section 2.2, were derived for networks with a flat one-level link sharing structure. In this section, we discuss how to generalize the delay bounds to networks in which links are hierarchically shared.

4.1 Directed Tree Model

First, we describe a directed tree model, largely borrowed from [1], for representation of a link sharing hierarchy. The root node, denoted by R, corresponds to the physical link, each leaf node corresponds to a flow with a queue of packets, and each non-leaf node (except for the root) corresponds to a link sharing entity, e.g., an administrative agency, a traffic type, or a service class. A non-leaf node n is called backlogged if at least one flow in its leaf descendent node set, denoted by $leaf(n)$, is backlogged. Conceptually, node n is a logical server for its descendents. The amount of work done by n in the time interval $[t_1, t_2]$ is defined to be $W_n(t_1, t_2) = \sum_{f \in leaf(n)} W_f(t_1, t_2)$, where $W_f(t_1, t_2)$ is the amount of flow f traffic served in the interval $[t_1, t_2]$. To achieve link sharing, each node m is assigned a weight $\phi_m > 0$, which is a relative measure of the link share desired by entity m. For ease of presentation and without loss of generality, we assume that

$$\begin{aligned} \phi_R &= 1 && (13) \\ \sum_{m \in children(n)} \phi_m &\leq \phi_n, \ \forall \text{ non-leaf node } n. && (14) \end{aligned}$$

Notation. Let $H(m)$ be the number of ancestors that node m has. Let $p^h(m)$ be the ancestor node of m that is h levels higher than m in the tree, $h = 1, 2, \ldots, H(m)$. Clearly $p^1(m) = parent(m)$ and $p^{H(m)}(m) = R$. For ease of presentation, we also set $p^0(m) = m$.

Hierarchical link sharing can have a big impact on the performance of flows that share the link. Specifically, packets of a flow under the link sharing hierarchy are scheduled jointly by *all* ancestor nodes of the flow before being served by the link [1]. For hierarchical link sharing to be useful, appropriate scheduling algorithms must be used at the logical servers to minimize any negative impact on flow performance. We next define ideal link sharing to set a performance target for designing such scheduling algorithms. Assume that the link bandwidth is C (bits/second).

Definition 1 *We say that ideal link sharing is achieved for logical server n if for any time interval $[t_1, t_2]$ in which n is continuously backlogged, the following holds*

$$W_n(t_1, t_2) \geq r_n \cdot (t_2 - t_1), \qquad (15)$$

where $r_n = \phi_n \cdot C$; that is, server n is guaranteed its link share whenever it has work to do.

It is straightforward to show that if every logical server is a Fluid model Fair Queueing (FFQ) server,[8] then ideal link sharing will be achieved for every logical server. Unfortunately, FFQ is not feasible. Therefore, scheduling algorithms that are good approximations of FFQ should be used at the logical servers.

[8] Also called General Processor Sharing (GPS) server in the literature [12].

4.2 Prior Work

In [1], Bennett and Zhang proposed a new approximation of FFQ, named Worst-case Fair Weighted Fair Queueing Plus (WF^{2}Q+), for providing tight delay bounds to flows under a link sharing hierarchy. They defined a metric for a logical server called Normalized Bit Worst-case Fair Index (NB-WFI).

Definition 2 *A logical server node n is said to guarantee a Normalized Bit Worst-case Fair Index (NB-WFI) of $\alpha_{m,n}$, for its child m, if during any time interval $[t_1, t_2]$ in which m is continuously backlogged, the following holds*

$$W_m(t_1, t_2) \geq \frac{\phi_m}{\phi_n} W_n(t_1, t_2) - \alpha_{m,n}. \qquad (16)$$

They then showed that delays of flow f packets are bounded by

$$\frac{\sigma_f}{r_f} + \sum_{h=0}^{H(f)-1} \frac{\alpha_{p^h(f)}}{r_{p^h(f)}}, \qquad (17)$$

where $\alpha_{p^h(f)}$ is the NB-WFI guaranteed to node $p^h(f)$ by node $p^{h+1}(f)$, if the flow is constrained by a leaky bucket (ρ_f, σ_f) with $\rho_f \leq r_f$. Based on this result, Bennett and Zhang claimed that small NB-WFI values for the logical servers are necessary for providing tight delay bounds to the flows. They also showed that among all proposed packet fair queueing servers, a WF^{2}Q+ server offers the smallest NB-WFI, namely: the length of one packet when a fixed packet length is used by all flows.

The problem of scheduling packets for flows under a link sharing hierarchy was also studied by Goyal et al. in [6], where an algorithm called Start-time Fair Queueing (SFQ) was proposed. Goyal et al. observed that the intermediate logical servers no longer have constant service rates for their children. Therefore, they analyzed the performance of SFQ in the context of Fluctuation Constrained servers first defined by Lee [10].

Definition 3 *A server is said to be Fluctuation Constrained (FC) with parameters*[9] *(C, δ), or simply a $FC(C, \delta)$ server, 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:*

$$W(t_1, t_2) \geq C \cdot (t_2 - t_1) - \delta. \qquad (18)$$

[9] A constant rate server is also FC with $\delta = 0$.

In addition, they showed that the service received by a flow from a FC SFQ server is also fluctuation constrained. Thus, if the logical servers (called virtual servers in [6]) are all SFQ servers, the packet delay bound for a flow under a link sharing hierarchy can be recursively computed. The exact bound is not given in [6].

4.3 A General Approach

While the authors cited above have made important contributions to the design of WF^{2}Q+ and SFQ, their approach is not the best. Specifically, too much emphasis was put on one good fair queueing algorithm for *both* link sharing and packet scheduling. For future networks, heterogeneous packet scheduling algorithms will likely be required at different parts of the link sharing hierarchy to achieve multiple design goals. For example, if implementation complexity is of primary concern, round robin scheduling may be more desirable than others. Also, there are a large number of performance results in the literature for one-level servers. It is not obvious from [1, 6] how these results, e.g. those for WFQ servers [2] or even FIFO servers, can be extended to networks with hierarchical link sharing.

Next we describe an approach in which link sharing and packet scheduling concerns are *separated.* In particular, a link sharing hierarchy is considered an extension of a one-level constant rate server.

Consider a particular flow f under the link sharing hierarchy. Even with hierarchical link sharing, the flow is in essence scheduled by a one-level server, its parent node, but with a variable service rate. The impact of all non-parent ancestors of f is indirectly accounted for by the service rate fluctuation of the parent node. Therefore, the analysis of flow f performance can be carried out in two steps:

(1) characterization of the service rate fluctuation of the parent server.

(2) extension of the performance results for one-level servers to account for service rate fluctuations (characterized in the previous step).

For step 1, we characterize the parent server, $n = parent(f)$, as a FC server. That is, there exists a constant $\delta_n \geq 0$ such that for any time interval $[t_1, t_2]$ in which n is backlogged (busy) throughout, the work (service) done by n satisfies

$$W_n(t_1, t_2) \geq r_n \cdot (t_2 - t_1) - \delta_n. \tag{19}$$

Note that the smaller δ_n is, the less service rate fluctuation for n. If all ancestors of n are FFQ servers, $\delta_n = 0$. Therefore, good approximations of FFQ should be used for ancestors of n to ensure a small value for δ_n. Let us look at two examples.

Example 1 *Assume that all ancestors of n are WF^{2}Q+ servers and all packets have a fixed length of L bits. From Definition 1 and the fact that WF^{2}Q+ guarantees a NB-WFI of L [1], it can be shown that δ_n is at most*

$$\left(\sum_{h=0}^{H(n)-1} \frac{r_n}{r_{p^h(n)}}\right) L. \tag{20}$$

Let us make the conservative assumption that $r_{p^h(n)}/r_{p^{h+1}(n)} = 0.5$, $h = 0, 1, \ldots, H(n)-1$. (The ratio is usually much smaller in reality.) Then it is easy to show that $\delta_n \leq 2L$.

Example 2 *Assume that all ancestors of n are SFQ servers and all packets have a fixed length of L bits. Applying Theorem 2 of [6], it can be shown that δ_n is at most*

$$\left(1 + \left(\sum_{h=0}^{H(n)-1} \frac{r_n}{r_{p^{h+1}(n)}}(Q(h)+1)\right)\right) L, \tag{21}$$

where $Q(h)$ is the number of branches that $p^{h+1}(n)$ has. Let us assume that $r_{p^h(n)}/r_{p^{h+1}(n)} = 0.5$ and $Q(h) = Q$, $h = 0, 1, \ldots, H(n)-1$. Then δ_n could be as large as $(Q+2)L$. Therefore in general, SFQ causes larger service rate fluctuations than WF^{2}Q+. On the other hand, SFQ appears to have a smaller implementation cost.

For step 2, we next show how to extend delay guarantee results from constant rate servers to FC servers.

Definition 4 *For a FC(C, δ) server, its* corresponding constant rate server *is defined to be identical to the FC server except that it has a constant service rate of C bits/second.*

Consider a class of *work conserving* servers, called the *priority class*, which can be described in general as follows. A priority value is computed and assigned to every packet upon its arrival, and queued packets are scheduled for service in the order of increasing priority values. Ties between packets are broken arbitrarily. Also, within a particular flow, the priority of each packet is non-decreasing in packet arrival time.

Notation. Consider an arbitrary sequence of packet arrivals to a priority server. (A packet arrival is represented by a tuple consisting of the packet arrival time and the packet size.) We use the following notation

$A(p)$	arrival time of packet p in the arrival sequence
$s(p)$	size of packet p (in bits)
$P(p)$	priority value assigned to packet p
$L(p)$	departure time of packet p

In what follows, we focus on a subclass called Service Independent Priority (SIP) servers, defined below.

Definition 5 *An $FC(C, \delta)$ priority server belongs to the Service Independent Priority (SIP) subclass if for every packet p of an arrival sequence to the server, $P(p)$ depends exclusively upon the value of C and packet arrivals up to and including p.*

Therefore for an $FC(C, \delta)$ SIP server, $P(p)$ is the same for different δ values.

It is easy to see that FC versions of Virtual Clock [15] and Delay-EDD [3] servers belong to the SIP class. FC versions of servers that approximate a hypothetical constant rate Fluid model Fair Queueing (FFQ) server (such as WFQ [2], and PGPS [12] and WF^2Q+ [1]) can be defined so that their packet priority values depend upon virtual times in the constant rate FFQ server and are thus SIP servers as well. FC FIFO servers also belong to the SIP class since they in effect use the arrival time of a packet as the packet priority. (Lee obtained some delay bound results for FC FIFO servers with leaky bucket constrained sources [10].)

In particular, we consider two types of SIP servers: *preemptive-resume with no overhead*, and *nonpreemptive*. For a SIP server that is preemptive-resume with no overhead, it will immediately stop the service of a packet and serve a newly arrived packet if the new arrival has a smaller priority value. But no work will be lost because of the preemption, i.e., when resuming service for the preempted packet, the server will start from where it stopped. For a SIP server that is *nonpreemptive*, the service of a packet cannot be preempted once it is started.

Next we present two theorems on extension of delay guarantees for SIP servers.[10] (Their proofs are in the appendix.) They deal with preemptive and nonpreemptive SIP servers respectively. We say that a delay guarantee has the *firewall* property if the guarantees to packets of a flow are independent of how other flows behave.

Theorem 1 *Consider an $FC(C, \delta)$ SIP server that is* preemptive-resume with no overhead*. If its corresponding constant rate server guarantees a departure deadline of $P(p) + \beta$ to every packet p of an arrival sequence, where β is a constant, and the guarantee has the firewall property and is independent of the priority tie breaking method, then it guarantees to every packet p a departure deadline of $P(p) + \beta + \frac{\delta}{C}$*

Corollary 1 *A $FC(C, \delta)$ FIFO server guarantees a delay bound of $\beta + \frac{\delta}{C}$ to every packet p of an arrival sequence if its corresponding constant rate server guarantees a delay bound of β to every packet p.*

[10]Note that the servers need to perform some form of admission control to provide meaningful delay guarantees.

For nonpreemptive servers, the service may be out-of-order sometimes. It happens when a newly arrived packet has a priority value smaller than that of the packet being served, but preemption is not allowed.

Definition 6 *Assume that a nonpreemptive server guarantees to packet p a deadline of $D(p)$. The guarantee is said to account for out-of-order service if with preemption the server can guarantee p a deadline of $D(p) - \dfrac{s_{max}}{C}$ where s_{max} is the maximum packet size.*

Theorem 2 *Consider a* nonpreemptive *$FC(C, \delta)$ SIP server. If its corresponding constant rate server guarantees a departure deadline of $P(p) + \beta$ to every packet p of an arrival sequence, and the guarantee has the firewall property, accounts for out-of-order service, and is independent of the priority tie breaking method, then it guarantees to every packet p a departure deadline of $P(p) + \beta + \frac{\delta}{C}$.*

Note that both theorems and Corollary 1 are quite general. They do not depend on specific admission control conditions or source control mechanisms. In contrast, most of Lee's analyses on FIFO FC servers [10] were done for leaky bucket constrained sources.

The burst (block) delay bounds presented in Section 2.2 were derived from a delay guarantee of Virtual Clock servers that has the firewall property and accounts for out-of-order service [14]. Using Theorem 2, it is straightforward to extend them to networks with hierarchical link sharing.

Corollary 2 *Consider the end-to-end burst delay bounds presented in Section 2.2. If at each switch k, the packets of the flow are served by a $FC(C_k, \delta_k)$ logical server instead of a constant rate link with capacity C_k, then the end-to-end delay of burst m of the flow is bounded as follows:*

$$D_m \leq \frac{b_m + g_m}{\lambda_m} + (K-1) \max_{1 \leq n \leq m} \{\frac{g_n}{\lambda_n}\} + \sum_{k=1}^{K} (\alpha_k + \frac{\delta_k}{C_k}) \quad (22)$$

$$D_m \geq (K-1)\frac{g_m}{\lambda_m} + \sum_{k=1}^{K} \alpha_k \quad (23)$$

Appendix

Proof of Theorem 1

Define

$N(p)$	the set of packets in the arrival sequence whose priority values are less than or equal to that of packet p
$W_p(t_1, t_2)$	total work (in bits) done by the FC server *for packets in* $N(p)$ in the time interval $[t_1, t_2]$

We will carry out a proof by contradiction. Specifically, we assume that there exists a packet p in the arrival sequence such that

$$L(p) > P(p) + \beta + \frac{\delta}{C}. \tag{24}$$

Then we will show

$$W_p(0, P(p) + \beta + \frac{\delta}{C}) \geq \sum_{q \in N(p)} s(q), \tag{25}$$

which contradicts (24).

We use the superscript c to label terms defined for the constant rate server. By definition of SIP servers, p has the same priority value, $P(p)$, in both server systems, and

$$N^c(p) = N(p). \tag{26}$$

From (24), there exists a time period, of which $[A(p), P(p) + \beta + \frac{\delta}{C}]$ is a sub-interval and in which the FC server is continuously busy with $N(p)$ packets. Let p^* be the packet that started this time period. (Note that p^* may be p itself.) In other words, there is no other $N(p)$ packet in the FC system when p^* arrived. Consider the set of packets in $N(p)$ that arrived prior to $A(p^*)$ in the FC system (same as the packets in $N^c(p)$ that arrived prior to $A(p^*)$ in the constant rate system). Since all of them have been served in the FC system by $A(p^*)$ (but not necessarily in the constant rate system), we have

$$W_p(0, A(p^*)) \geq W_p^c(0, A(p^*)). \tag{27}$$

Since the FC server is busy exclusively with packets in $N(p)$ throughout the interval $[A(p^*), P(p) + \beta + \frac{\delta}{C}]$, we have

$$\begin{aligned} & W_p(A(p^*), P(p) + \beta + \frac{\delta}{C}) \\ \geq \;& C \cdot (P(p) + \beta + \frac{\delta}{C} - A(p^*)) - \delta && (28) \\ = \;& C \cdot (P(p) + \beta - A(p^*)) && (29) \\ = \;& W^c(A(p^*), P(p) + \beta) && (30) \end{aligned}$$

Combining (27) with (30), we have

$$\begin{aligned} & W_p(0, P(p) + \beta + \frac{\delta}{C}) \\ \geq \;& W_p^c(0, A(p^*)) + W^c(A(p^*), P(p) + \beta) && (31) \end{aligned}$$

Consider the constant rate server and a modification to the arrival sequence as follows. Those packets in $N^c(p)$ that arrive at or after $A(p) + \dfrac{s(p)}{C}$ in the original sequence will have their arrival times moved forward in the modified sequence such that they all arrive in the time interval $(A(p), A(p) + \dfrac{s(p)}{C})$ and the original order of arrivals for each flow is preserved. Note that the priority value of p as well as the guaranteed deadline of p are unaffected by the modification. Moreover, since the priority value of each packet within a flow is non-decreasing in packet arrival time (by definition of a priority class server), $N^c(p)$ is unaffected by the modification as well.

The deadline guarantee of $P(p) + \beta$ by the constant rate server still holds for the modified packet arrival sequence because of the firewall property. Moreover the guarantee is independent of the tie breaking method. Therefore it would hold even if p were served last among all $N^c(p)$ packets. Also with the modified arrival sequence, the most amount of work that the constant rate server can do for $N^c(p)$ packets in the time interval $[0, P(p) + \beta)]$ is: $W_p^c(0, A(p^*)) + W^c(A(p^*), P(p) + \beta)$. Therefore, the following must hold

$$W_p^c(0, A(p^*)) + W^c(A(p^*), P(p) + \beta) \geq \sum_{q \in N^c(p)} s(q). \tag{32}$$

Combining (31) and (32), we have

$$W_p(0, P(p) + \beta + \frac{\delta}{C}) \geq \sum_{q \in N^c(p)} s(q). \tag{33}$$

Because of (26), $N^c(p)$ can be substituted by $N(p)$ in (33). Therefore (25) holds. □

Proof of Corollary 1

For FIFO servers, $P(p) = A(p)$ for all p. Therefore a delay bound guarantee of β is equivalent to a deadline guarantee of $P(p) + \beta$. The proof for the deadline guarantee generalization is identical to the one for Theorem 1 except for a simpler reasoning for (32): While a deadline guarantee by a FIFO server does not have the firewall property, there is no need for the condition because all packets in $N^c(p)$ arrive no later than $A(p)$; therefore we can reason for (32) with the original arrival sequence. □

Proof of Theorem 2

Again we carry out a proof by contradiction. Specifically, we assume that for the nonpreemptive FC server, there exists a packet p in the arrival sequence such that

$$L(p) > P(p) + \beta + \frac{\delta}{C}. \tag{34}$$

Then we will show

$$W_p(0, P(p) + \beta + \frac{\delta}{C}) \geq \sum_{q \in N(p)} s(q), \tag{35}$$

which contradicts (34).

In this proof, we also consider a third SIP server, which is identical to the constant rate server except that it is preemptive-resume with no overhead. We use the superscripts, c and pre, to label terms for respectively the nonpreemptive and preemptive constant rate server systems. Note that by definition of SIP servers, p has the same priority value, $P(p)$, for all three server systems, i.e.,

$$N^{pre}(p) = N^c(p) = N(p). \tag{36}$$

It is given that

$$L^c(p) \leq P(p) + \beta, \tag{37}$$

and the guarantee in (37) accounts for out-of-order service. Therefore we have

$$L^{pre}(p) \leq P(p) + \beta - \frac{s_{max}}{C}. \tag{38}$$

Note that the above guarantee also has the firewall property.

Similar to the proof for Theorem 1, we can show that there exists a packet $p^* \in N(p)$ such that

$$W_p(0, A(p^*)) \geq W_p^{pre}(0, A(p^*)), \tag{39}$$

$$W_p(A(p^*), P(p) + \beta + \frac{\delta}{C}) \geq C \cdot (P(p) + \beta - A(p^*)) - s_{max}. \tag{40}$$

The extra term s_{max} in (40) is due to the fact that a part of the work done by the nonpreemptive FC server in $[A(p^*), P(p)+\beta+\frac{\delta}{C}]$ may be for out-of-order service of a packet that is not in $N(p)$.

Similar to the proof for Theorem 1, we have

$$W_p^{pre}(0, A(p^*)) + W^{pre}(A(p^*), P(p) + \beta - \frac{s_{max}}{C}) \geq \sum_{q \in N^{pre}(p)} s(q). \tag{41}$$

Moreover

$$\begin{aligned} & W^{pre}(A(p^*), P(p) + \beta - \frac{s_{max}}{C}) \\ &= C \cdot (P(p) + \beta - \frac{s_{max}}{C} - A(p^*)) \quad (42) \\ &= C \cdot (P(p) + \beta - A(p^*)) - s_{max}. \quad (43) \end{aligned}$$

Combining (41) and (43), we have

$$W_p^{pre}(0, A(p^*)) + C \cdot (P(p) + \beta - A(p^*)) - s_{max} \geq \sum_{q \in N^{pre}(p)} s(q). \tag{44}$$

From (39), (40), (41) and (44), we have

$$\begin{aligned} & W_p(0, P(p) + \beta + \frac{\delta}{C}) \\ &= W_p(0, A(p^*)) + W_p(A(p^*), P(p) + \beta + \frac{\delta}{C}) \quad (45) \\ &\geq W_p^{pre}(0, A(p^*)) + C \cdot (P(p) + \beta - A(q)) - s_{max} \quad (46) \\ &\geq \sum_{q \in N^{pre}(p)} s(q). \quad (47) \end{aligned}$$

Because of (36), $N^{pre}(p)$ can be substituted by $N(p)$ in (47). Therefore (35) holds. □

References

[1] Jon C.R. Bennett and Hui Zhang. Hierarchical packet fair queueing algorithms. In *Proceedings of ACM SIGCOMM '96*, pages 143–156, Stanford University, CA, August 1996.

[2] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simulation of a fair queuing algorithm. In *Proceedings of ACM SIGCOMM '89*, pages 3–12, August 1989.

[3] D. Ferrari and D. Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE JSAC*, pages 368–379, April 1990.

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

[5] The ATM Forum. *ATM Traffic Management Specification.* 1995. Version 4.0.

[6] P. Goyal, H. M. Vin, and H. Cheng. Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks. In *Proceedings of ACM SIGCOMM '96*, pages 157–168, Stanford University, CA, August 1996.

[7] ITU-T Rec. I.371. *Traffic Control and Congestion Control in B-ISDN.* Perth, U.K., Nov. 6-14, 1995.

[8] Simon S. Lam and Geoffrey G. Xie. Burst Scheduling networks. Technical Report TR-94-20, University of Texas at Austin, Austin, Texas, July 1994. Revised, August 1996. An abbreviated version in *Proceedings of IEEE INFOCOM '95*, April 1995.

[9] Simon S. Lam and Geoffrey G. Xie. Group priority scheduling. Technical Report TR-95-28, University of Texas at Austin, Austin, Texas, July 1995. Revised, September 1996. An abbreviated version in *Proceedings of IEEE INFOCOM '96*, March 1995.

[10] Kam Lee. Performance bounds in communication networks with variable-rate links. In *Proceedings of ACM SIGCOMM '95*, pages 126–137, August 1995.

[11] Michel Loève. *Probability Theory I.* New York, 1977. 4th ed.

[12] Abhay K. Parekh and Robert G. Gallager. A generialized processor sharing approach to flow control in integrated services networks: The single node case. *IEEE/ACM Trans. on Networking*, 1(3):344–357, June 1993.

[13] A. Romanow and S. Floyd. The dynamics of TCP traffic over ATM networks. In *Proceedings of ACM SIGCOMM '94*, London, England, August 1994.

[14] Geoffrey G. Xie and Simon S. Lam. Delay guarantee of Virtual Clock server. *IEEE/ACM Trans. on Networking*, 3(6):683–689, December 1995.

[15] Lixia Zhang. VirtualClock: A new traffic control algorithm for packet switching networks. In *Proceedings of ACM SIGCOMM '90*, pages 19–29, August 1990.