# A Fair Leaky-Bucket Traffic Shaper for ATM Networks

Jennifer L. Rexford
University of Michigan*

Albert G. Greenberg
AT&T Bell Laboratories

Flavio G. Bonomi
AT&T Platform Organization, AT&T Network Systems

**Abstract**

ATM network performance hinges on user sessions, or *virtual connections*, adhering to the traffic contracts established by the admission control policy. This paper considers efficient mechanisms for enforcing these contracts by *shaping* the incoming cell streams. While shaping effectively accommodates variation in a connection's traffic flow, this smoothing function introduces implementation complexity, since the shaper must buffer violating cells and schedule them for later transmission. Conflicts arise when multiple cells, from different connections, become eligible for transmission at the same time. This paper presents a *fair leaky-bucket* (FLB) shaper that minimizes the traffic distortions caused by these cell collisions. A theorem shows that FLB shaping, based on *weighted fair queueing*, closely preserves connection leaky-bucket parameters and bounds cell shaping delay. The paper also presents an efficient implementation of *self-clocked fair queueing*, which reduces the complexity of the FLB architecture. Finally, simulation experiments demonstrate how FLB shaping mitigates the harmful effects of collisions between competing cells.

## 1 Introduction

Emerging high-speed Asynchronous Transfer Mode (ATM) networks must support a wide range of applications with different traffic characteristics and performance requirements. Admission control policies govern whether the network can accommodate a new session, or virtual connection, based on its traffic descriptors and quality-of-service requirements. Network performance strongly depends on the characteristics of the traffic being multiplexed on the links. Consequently, admission control policies for ATM networks rely on the admitted connections complying with their traffic contract as their cells enter and travel through a network. Also, compliance with traffic contracts is fundamental whenever cells from a single connection are routed through switches with possibly different performance characteristics, or when cells cross the boundaries of network domains belonging to different service providers.

Three types of mechanisms have been proposed to administer such contracts:

*Jennifer Rexford's work on this project was supported by the AT&T Graduate Research Program for Women.

- mechanisms for enforcing a connection's traffic agreements by *policing* or *shaping* its traffic at the *ingress* of a network;
- *fair scheduling* mechanisms which preserve, to the extent possible, a connection's traffic descriptors, as its cells travel through a network;
- mechanisms for *shaping* a connection's traffic at the *egress* of a network to enforce compliance with a traffic contract with a downstream network.

To support their implementation at high speed, the mechanisms must incur a small amount of work per cell. In this paper, we propose a *fair shaping* mechanism which combines the actions of shaping and fair scheduling. This mechanism can efficiently provide policing and shaping at ATM switch ingress ports, as well as fair scheduling and shaping at switch egress ports.

A network can employ traffic *policing* to monitor and regulate accepted connections (see Figure 1). When an arriving cell violates its connection's contract, a network policer either discards the "non-conforming" cell or marks the cell so as to assign it to a low priority class. This prevents malicious or heavily-loaded connections from compromising the performance of other connections, and significantly improves the network's ability to predict and guarantee each connection's quality of service.

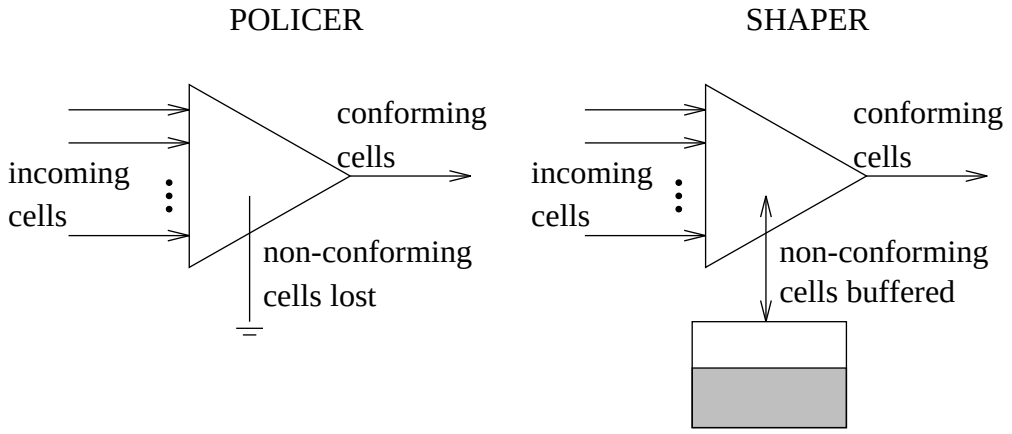



Figure 1: Policer and Shaper Multiplexors. Cells that do not violate a connection's contract are termed "conforming" and can depart immediately; other cells are termed "non-conforming."

If a connection momentarily exceeds its traffic parameters, a policer may drop or mark several cells, even if the connection obeys its traffic contract over a larger time interval. Instead of dropping or marking non-conforming traffic, a *shaper* delays cells until they conform to the connection's traffic descriptors, which typically prescribe burst and bandwidth restrictions. Shaping accommodates variation in traffic flow by smoothing the incoming cell stream. Without such buffering, the network must book more resources to accommodate bursts or the session must suffer loss or settle for a lower rate connection. In addition, a shaper can smooth traffic traveling between networks, which may

operate with different switches, possibly at different link speeds. Although traffic shaping can reduce cell loss, the smoothing function introduces additional delay and implementation complexity, since the shaper must buffer non-conforming cells and schedule them for later transmission. A single network-access point often services *thousands* of connections, with different traffic parameters, requiring efficient and scalable techniques for buffering and scheduling cells.

Many existing shapers and policers employ some version of leaky-bucket control to enforce burst and bandwidth restrictions on the incoming cell stream [1–3]. A leaky-bucket controller generates credit tokens at rate $\rho$, where the token bucket holds at most $\sigma$ credits; an arriving cell must claim a token before entering the network. Given the status of the token bucket for each connection, the shaper can determine the conformance time of an arriving cell. A connection is said to be $(\sigma, \rho)$-compliant if all of its cells are conforming [4].

Though appropriate leaky-bucket type controls can ensure stable behavior over a long time scale, these controls do not in themselves mitigate critical conflicts that may arise over quite short time intervals. Ideally, the shaper transmits a cell as soon as it conforms to the connection traffic descriptors. Conflicts occur when multiple cells, from different connections, become eligible for transmission during the same time slot. Depending on how the shaper arbitrates amongst competing cells, these collisions can distort the leaky-bucket parameters and increase cell shaping delays, even for conforming traffic. These traffic distortions violate the expectations of downstream nodes in the network, possibly increasing cell loss rates and end-to-end delay. As shown in the next section, FIFO servicing of conforming traffic cannot prevent these harmful interactions.

Fair queueing schemes, such as weighted fair queueing [5–7] and self-clocked fair queueing [8,9], guarantee that each connection receives its share of the link bandwidth on a small time scale. Parekh and Gallager [6] have shown that if a fair queueing server handles traffic that is already leaky-bucket compliant, the fair arbitration can closely preserve connections' $\sigma$ and $\rho$ parameters. However, if the traffic may be non-compliant and non-conforming cells are buffered, then new challenges arise in designing a mechanism, scalable to thousands of sessions, that *integrates* centralized leaky-bucket controls with fair queueing.

The first design that comes to mind might consist of a staging area for non-conforming traffic, followed by a fair queueing server that arbitrates among cells that have reached their conformance times. However, it appears very difficult to scale this design to accommodate thousands of sessions. As illustrated below, a very large number buffered cells can become conforming simultaneously, and it is problematic how to shift this mass of cells to the fair queueing server while maintaining fair treatment of future arrivals. In the *fair leaky bucket* (FLB) shaper proposed here, this implementation complexity is avoided by allowing *all* arriving cells to proceed directly to the fair queueing server; if the server selects a non-conforming cell for transmission, the shaper reschedules the cell for later service.

The remainder of the paper is organized as follows. Section 2 discusses shaper architectures

and motivates fair traffic shaping. In Section 3, we prove that FLB shaping, based on weighted fair queueing, closely preserves connection leaky-bucket descriptors as long as connections do not have widely different bandwidth requirements. Section 4 describes an efficient implementation of the FLB shaper, based on self-clocked fair queueing. In Section 5, we present simulation results that show how the FLB shaper bounds traffic distortions. Section 6 concludes the paper with a discussion of future research directions in fair traffic shaping.

The FLB shaper complements related research on architectures for traffic shaping in ATM networks [10–15], discussed further in Section 2. Parekh and Gallager's analysis [6, 7] of weighted fair queueing, also known as packet-by-packet generalized processor sharing, provides the theoretical underpinning for the theorem in Section 3. They show that weighted fair queueing closely preserves the traffic descriptors of connections that are already leaky-bucket compliant; the FLB shaper extends these results to efficiently convert non-conforming traffic into a compliant output stream. The implementation approach in Section 4 relates to other mechanisms for efficient rate control in high-speed networks [16–19, 15].

# 2 Motivation

## 2.1 Shaper Implementation

An efficient shaper architecture must scale well with the number and type of connections. A shaper can maintain a separate controller and buffer for each connection, but a scalable implementation requires a more integrated approach. Scalable architectures typically maintain a table that holds the shaping status of each connection [11–13]. When a new cell arrives, the shaper identifies the appropriate connection and updates its state, based on the elapsed time since the last update. (To avoid clock wrap-around errors, the shaper can periodically update each connection's state, independent of cell arrivals [13].) The shaper uses this state to determine the time an arriving cell first conforms to its connection's traffic descriptors. The shaper uses these conformance times to schedule cells for transmission.

Traffic shapers in high-speed ATM networks necessitate simple and efficient techniques for recognizing when cells conform to their traffic descriptors. The most general approach uses a hardware priority queue to rank the cells in order of increasing conformance times [11, 10]; during each transmission slot, the shaper transmits the queue's head cell, if its conformance time has been reached. To avoid the implementation complexity of a hardware priority queue, the Spacer-Controller dynamically builds a link transmission schedule using a large collection of bins, with one bin for each time slot [12]; an arriving cell locates the first empty bin at or after its conformance time.

An alternate implementation constructs a linked list of cells in each bin; the shaper transmits all cells in one list before proceeding to the next non-empty list in the schedule [14]. Although these

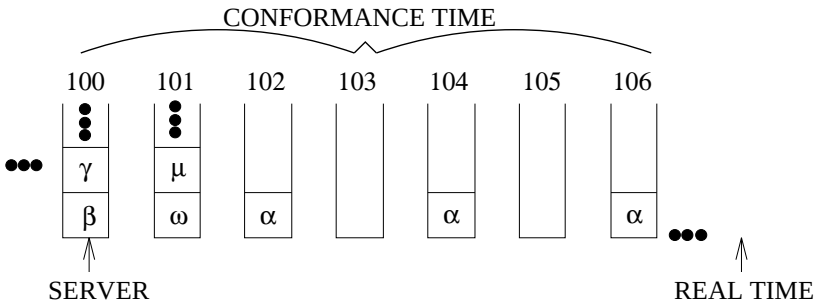



Figure 2: Example of cell collisions in a traffic shaper

discrete-time bins obviate the need for a hardware priority queue, a shaper implementation requires a large number of these bins to handle cells with conformance times far into the future. The shaper can reduce the number of bins by having each bin represent a *range* of conformance times [15], but such coarse-grain scheduling can distort the outgoing traffic, particularly for high-bandwidth connections.

## 2.2 Traffic Distortions

When multiple cells become conforming at the same time, a shaper architecture must arbitrate amongst the backlogged conforming cells. Figure 2 illustrates a class of examples, where shaping can lead to an egregious burst of cells, and an important associated scheduling problem. It is not difficult to construct such examples even for ingress shaping, where the number of input lines into the shaper is limited and the speed of the shaper's output line equals the sum of the speeds of its input lines. In the example, a backlog of cells accumulates over time due to the arrival of non-conforming cells (from connections $\beta$, $\gamma$, $\mu$, $\omega$,...); these backlogged cells all become conforming nearby in time (times $100, 101$). Suppose further that other connections (connection $\alpha$) submit conforming cells near the conformance times for the backlogged connections (time $102, 104, 106, \ldots$).

Since the link transmits at most one cell in each time slot, the shaper falls behind in servicing the backlogged traffic. Thus, real time advances far beyond time 100 while the shaper drains the traffic from times 100 and 101. By the time the shaper finishes sending these cells, a high-bandwidth connection $\alpha$ may have several conforming cells awaiting service, causing the link to transmit a large burst of traffic from this connection. Though connection $\alpha$'s cells may have arrived beautifully spaced, the shaper essentially slams them together on the output link. FIFO service of conforming cells cannot avoid this problem. Instead, a *fair* shaper could transmit some of the cells from connection $\alpha$ *before* sending all of the cells that reached their conformance times earlier. This would avoid producing an excessive burst of connection $\alpha$'s cells by interleaving this traffic with cells from other connections.

## 2.3 Fair Queueing

Collisions between conforming cells arise because multiple connections compete for access to the outgoing link. A shaper could preserve connection leaky-bucket parameters if each connection $\alpha$ had access to a *private* link of rate at least $\rho_\alpha$, its average bandwidth requirement, instead of sharing a link with other traffic. While this is not practically feasible, it provides a useful paradigm for governing the interaction between competing connections. *Generalized processor sharing* (GPS) models this abstraction by continuously dividing link bandwidth amongst the backlogged connections, in proportion to their weights $r_\alpha$ [5,6]. If these weights are chosen as $r_\omega \propto \rho_\omega$, and $\sum_\omega \rho_\omega < 1$, the link can continuously serve connection $\alpha$ cells at rate $\rho_\alpha$, or faster. This ensures that GPS preserves the leaky-bucket parameters for each connection [6,7].

GPS cannot be directly implemented, since it requires preemption of the link resource on an arbitrarily small time scale. However, packetized versions retain the fairness properties of this idealized multiplexing model. Weighted fair queueing (WFQ) approximates the fluid model by ranking cells by the time they would complete service under GPS [5]. In effect, then, a WFQ server *simulates* GPS to order the cells awaiting access to the link. WFQ simulates GPS by maintaining a virtual time $R(t)$ such that $R(0) = 0$ and

$$\frac{dR(t)}{dt} = R'(t) = \left( \sum_{\omega \in B(t)} r_\omega \right)^{-1}$$

where $B(t)$ is the set of busy connections in the GPS simulation at time $t$. The logical clock $R(t)$ represents the GPS server's progress in servicing the backlogged connections, while $R'(t)$ is the normalized service rate.

Maintaining $R(t)$ allows the GPS simulation to determine the virtual starting and finishing times for an incoming cell, independent of future arrivals to the system [5]. If cell $k$ of connection $\alpha$ enters the system at time $t$, its virtual starting and finishing times under GPS service are

$$\begin{aligned} S_k^\alpha &= \max\{F_{k-1}^\alpha, R(t)\} \\ F_k^\alpha &= S_k^\alpha + 1/r_\alpha, \end{aligned}$$

assuming a common cell length 1, as in ATM. If cell $k$ arrives to an empty connection queue, GPS begins serving the cell as soon as it arrives (at virtual time $R(t)$); otherwise, the cell enters service when GPS finishes transmitting the previous cell from the connection (at virtual time $F_{k-1}^\alpha$). The $1/r_\alpha$ term represents the normalized work required to service the new cell. When WFQ must select a cell for transmission, the server picks the cell with the smallest virtual finishing time $F$, among the cells already queued for service (breaking ties arbitrarily).

Computing $R(t)$ is complex, and indeed is similar to simulating multiple GPS service events between consecutive cell departures [20,21]. Self-clocked fair queueing (SCFQ) [8,9] uses the virtual

finishing time $F^{serv}$ of the cell *currently in service* as an estimate of current virtual time, thereby avoiding the burden of computing $R'(t)$. Analysis [9] and simulation [8] suggest that SCFQ retains the throughput fairness properties of GPS and WFQ, while simplifying the assignment of virtual finishing times. In addition, a SCFQ server can postpone assigning a cell's $F$ value until the cell reaches the head of its connection's queue, since virtual finishing times do not depend on the status of an underlying GPS queue; when a cell becomes the head of its connection queue, its virtual finishing time is $F^{\alpha} = F^{serv} + 1/r_{\alpha}$, whether the cell arrives to an idle connection or ascends to the head position when the previous cell departs. The FLB architecture capitalizes on these properties to efficiently shape traffic in ATM networks.

# 3 Fair Leaky-Bucket Shaping

WFQ and SCFQ closely preserve connection leaky-bucket descriptors by arbitrating fairly amongst competing connections. These properties suggest that fair queueing algorithms can also play a useful role in shaping *non-conforming* incoming traffic into a compliant output stream.

## 3.1 Fair Shaping

The fair leaky-bucket (FLB) shaper combines leaky-bucket control with fair queueing algorithms to smooth the incoming traffic and limit distortion in the resulting output streams. The FLB shaper reduces implementation complexity by allowing *all* arriving cells to proceed directly to the fair queueing server. Admitting non-conforming traffic to the fair queue avoids complex hardware mechanisms for detecting newly conforming cells and submitting them to the server. Hence, the FLB shaper optimistically assigns a virtual finishing time to a connection's head-of-line cell, even if this cell has not reached its conformance time yet. Although this significantly reduces implementation complexity, it could cause the fair queueing arbiter to select a non-conforming cell for service.

Figure 3 shows how the FLB shaper handles cell arrivals and cell selection by the fair queueing server. When cell $k$ arrives to connection $\alpha$, the FLB shaper determines the cell's conformance time $t_c^{\alpha,k}$ and appends this cell to the connection $\alpha$ queue, as shown in Figure 3(a). If this connection was previously idle, the server assigns the cell's virtual finishing time $F^{\alpha}$. At the start of each transmission slot, the fair queueing server selects the cell with the smallest virtual finishing time, as shown in Figure 3(b). While the link can service a conforming cell, transmitting non-conforming traffic would violate connection leaky-bucket parameters. Instead, the FLB shaper treats this non-conforming cell as a new arrival to the *head* of its connection queue by assigning it a new virtual finishing time. In effect, the server *reschedules* the non-conforming cell, after accounting for the link bandwidth the cell *would* have consumed.

This ensures that resubmitting a non-conforming cell does not penalize the traffic in other

connections. Although rescheduling the cell consumes a transmission slot, even if other connections await service, WFQ and SCFQ guarantee that each connection receives sufficient link bandwidth, independent of other traffic. A resubmission can only occur when a connection has no leaky-bucket tokens and no conforming cells awaiting service. Hence, the affected connection has been receiving good service from the shaper, since this non-conforming cell has not become conforming since the transmission of the connection's previous cell.

### 3.2 Preservation of Leaky-Bucket Parameters

Theorem 2 below provides a worst-case bound on the distortion to the burstiness parameter $\sigma$ of a $(\sigma, \rho)$-compliant session brought on by rescheduling non-conforming cells. As noted in Section 1, this theorem assumes weighted fair queueing arbitration in the FLB shaper; we expect a similar result to hold for self-clocked fair queueing. Extending the argument to other fair queueing schemes requires counterparts to (2) and (3), below, which provide upper and lower bounds for the throughput discrepancy between generalized processor sharing and the packetized fair queueing scheme.

As a first step, we adapt a basic result of Parekh and Gallager [6] on leaky-bucket control.

**Lemma 1** *Consider a system of connections arriving to a multiplexor with an unbounded buffer, where connection $\alpha$ has leaky-bucket parameters $(\sigma_\alpha, \rho_\alpha)$. While connection $\alpha$'s cells need not conform to the leaky-bucket descriptors on arrival, the multiplexor transmits only conforming traffic. Connection $\alpha$'s output stream is $(\sigma_\alpha^{out}, \rho_\alpha)$-compliant where*

$$\sigma_\alpha^{out} \leq q_\alpha^*, \tag{1}$$

*where $q_\alpha^*$ is the maximum possible backlog of conforming cells for connection $\alpha$.*

**Proof.** Let $q_\alpha(t)$ denote the amount of *conforming* traffic from connection $\alpha$ that is buffered at time $t$; the connection may also have other, non-conforming traffic awaiting service. Also, let $\ell_\alpha(t)$ denote the number of connection $\alpha$ tokens available at time $t$. For any time interval $[s, t)$, let

```
Cell k arrives for connection α;
Determine conformance time t_c^{α,k};
if (connection α queue is empty)
   F^α = F^serv + 1/r_α;
Add cell k to connection α queue;
```

(a) Cell arrival

```
Find connection α with smallest F^α;
if (t_c^{α,k} ≤ t)
   transmit cell;
if (connection α queue is non-empty)
   F^α = F^serv + 1/r_α;
```

(b) Cell selection

Figure 3: Fair leaky-bucket algorithm under self-clocked fair queueing

$a_\alpha(s,t)$ denote the number of connection $\alpha$ cells that become conforming during $[s,t)$, including any conforming traffic that arrives during the interval. Let $w_\alpha(s,t)$ denote the number of connection $\alpha$ cells transmitted during this interval. (These quantities may be fractional.) Since the multiplexor transmits only conforming traffic, conservation of cells requires

$$w_\alpha(s,t) = a_\alpha(s,t) + q_\alpha(s) - q_\alpha(t).$$

The leaky-bucket parameters $(\sigma_\alpha, \rho_\alpha)$ ensure that $a_\alpha(s,t) \leq \ell_\alpha(s) + \rho_\alpha(t-s)$, so

$$w_\alpha(s,t) \leq (\ell_\alpha(s) + q_\alpha(s) - q_\alpha(t)) + \rho_\alpha(t-s),$$

which implies that $\sigma_\alpha^{out} \leq \ell_\alpha(s) + q_\alpha(s) - q_\alpha(t) \leq \ell_\alpha(s) + q_\alpha(s)$. Finally, note that $\ell_\alpha(s) + q_\alpha(s)$ cannot exceed $q_\alpha^*$, since this would allow the connection to instantaneously increase $q_\alpha^*$ by consuming all $\ell_\alpha(s)$ remaining tokens at time $s$. □

To analyze the FLB shaper, we draw on two results relating the throughputs of WFQ and GPS systems, for connections with *arbitrary* traffic characteristics. Consider a system of $N$ connections flowing into a multiplexor with an unbounded cell buffer, where each connection $\omega$ has positive weight $r_\omega$. Let $W_\alpha^{WFQ}(s,t)$ denote the amount of connection $\alpha$ traffic served during interval $[s,t)$ by a WFQ server; similarly, let $W_\alpha^{GPS}(s,t)$ denote the total service given by a GPS server. In particular, under FLB shaping, $W_\alpha$ refers to *all* work performed in connection $\alpha$'s behalf, including any time slots spent rescheduling non-conforming cells. Parekh and Gallager [6] proved that WFQ lags at most one cell behind GPS in serving a connection. That is, for all $\alpha$, $s$, and $t \geq s$,

$$W_\alpha^{GPS}(s,t) - W_\alpha^{WFQ}(s,t) \leq 1. \tag{2}$$

The performance of the FLB shaper depends on this bound, as well as a new throughput bound in the reverse direction, proven in Appendix A. In particular, for all $\alpha$, $s$, and $t \geq s$,

$$W_\alpha^{WFQ}(s,t) - W_\alpha^{GPS}(s,t) \leq \min\{N-1, r_\alpha/r_{\min}\}, \tag{3}$$

where $r_{\min} = \min_\omega\{r_\omega\}$ denotes the smallest weight. It can be shown that both (2) and (3) are tight bounds. Now, we are in position to prove the main result on the performance of the FLB shaper.

**Theorem 2** *Consider a system of multiple connections, where connection $\alpha$ traffic is shaped using leaky-bucket parameters $(\sigma_\alpha, \rho_\alpha)$, and $\sum_\alpha \rho_\alpha < 1$. FLB shaping, with WFQ arbitration and weights $r_\alpha \propto \rho_\alpha$, guarantees that the corresponding output stream satisfies $(\sigma_\alpha^{out}, \rho_\alpha)$, where*

$$\sigma_\alpha^{out} \leq \max\{\sigma_\alpha + 1, \min\{N+1, 2 + \rho_\alpha/\rho_{\min}\}\}.$$

**Proof.** The proof employs a reference GPS system that sees the same traffic as the FLB WFQ system. As constructed below, this reference system sees the same arrivals as the underlying GPS simulation that determines virtual finishing times, up to a renaming of non-conforming cells that are rescheduled. For each cell that arrives to the FLB shaper, a copy of the cell arrives to the GPS system at the same time. Moreover, for each non-conforming cell that the FLB shaper reschedules at a given time $t$, a copy of that cell also arrives to the GPS system at time $t$. Thus, the GPS system is driven by the original arrivals and the rescheduling actions of the FLB shaper, and attempts no rescheduling itself. To construct the arrival sequence for GPS, we can imagine running the FLB system from time 0 to $\infty$, and marking all of the arrival and rescheduling events for each connection. All these events are arrival events for the GPS system. Let us refer to the rescheduling events as *clone* arrivals.

These arrivals to the GPS server determine the *sequence* of virtual finishing times for each connection. However, FLB shaping schedules clone arrivals at the *head* of the connection queue. Still, this does not alter the sequence of virtual finishing times generated by the underlying GPS simulation. Hence, the FLB WFQ server uses these sequences of virtual finishing times to arbitrate amongst competing connection, but services the cells from an individual connection in order of conformance times. Focus on one connection $\alpha$. At a given time $t$, let $Q_\alpha^{GPS}(t)$ denote the amount of traffic (including non-conforming and clone cells) buffered for connection $\alpha$ in the GPS system; similarly, let $Q_\alpha^{FLB}(t)$ denote the number of buffered cells in the FLB WFQ system. It follows from the construction of the arrival sequence and the throughput bounds of (2) and (3) that, for all $t$,

$$Q_\alpha^{FLB}(t) - 1 \leq Q_\alpha^{GPS}(t) \leq Q_\alpha^{FLB}(t) + r_\alpha / r_{\min}.$$

A similar result holds for the *conforming* traffic,

$$q_\alpha^{FLB}(t) - 1 \leq q_\alpha^{GPS}(t) \leq q_\alpha^{FLB}(t) + r_\alpha / r_{\min}, \tag{4}$$

since the two systems experience the same cell arrival times (including clone arrivals) and serve connection $\alpha$'s cells in the same order (based on their conformance times).

We argue below that, for all $t$,

$$q_\alpha^{GPS}(t) \leq \max\{\sigma_\alpha, \min\{N, 1 + r_\alpha / r_{\min}\}\}, \tag{5}$$

implying $q_\alpha^{FLB}(t) \leq \max\{\sigma_\alpha + 1, \min\{N+1, 2 + r_\alpha / r_{\min}\}\}$ for all $t$. Applying Lemma 1 then completes the proof; this finite bound on queue length also implies that rescheduling non-conforming cells does not violate system stability. Hence, it remains then to prove (5) for the GPS system. Initially the system is empty; i.e., $q_\alpha^{GPS}(0) = 0$. We show that the bound of (5) holds until a time where $q_\alpha^{GPS}$ reaches zero again; however, if $q_\alpha^{GPS}$ returns to zero, then the same argument and bound applies to its next excursion from zero. By a straightforward induction, the bound must always hold.

Consider the connection $\alpha$ cells arriving to the GPS system. Until the first clone arrives, the system serves only $(\sigma_\alpha, \rho_\alpha)$-conforming cells from connection $\alpha$; thus, $q_\alpha^{GPS}(t) \leq \sigma_\alpha$ in this interval, since GPS continuously serves connection $\alpha$ at a rate of *at least* $\rho_\alpha$ (since $r_\omega \propto \rho_\omega$ for all $\omega$) [7]. Now, consider the first clone arrival. Under FLB shaping, connection $\alpha$'s cell is rescheduled only if the connection's token bucket is empty, $q_\alpha^{FLB} = 0$, and the WFQ server elects to serve connection $\alpha$. The shaper may resubmit this cell one or more times, until the cell finally becomes conforming at time $t_c^\alpha$.

At $t_c^\alpha$, then, $q_\alpha^{FLB}$ increases to 1, since connection $\alpha$ now has one conforming cell awaiting service. By (4), this implies that $q_\alpha^{GPS} \leq \min\{N, 1 + r_\alpha/r_{\min}\}$, where the GPS server's buffered traffic may include clones that have not been served yet. Under GPS, the server will now continuously service the backlogged connection $\alpha$ at a rate of *at least* $\rho_\alpha$, while the connection generates new conforming cells at a rate of *at most* $\rho_\alpha$. Hence, GPS never has a backlog of more than $\min\{N, 1 + r_\alpha/r_{\min}\}$ conforming cells, until $q_\alpha^{GPS}$ returns to zero. □

This bound on queue length also implies a bound on cell shaping delay for conforming traffic. When cell $k$ from connection $\alpha$ becomes conforming at time $t$, any previous cells from that connection are also conforming. Hence, connection $\alpha$ will not experience any new clone arrivals before cell $k$ departs the shaper, and any backlogged clone cells have become conforming. At time $t$, there are a bounded number of conforming cells (including clones) ahead of cell $k$ in the underlying GPS queue, due to (5). GPS serves this connection at a rate of at least $\rho_\alpha$, so cell $k$ incurs at delay of at most

$$\frac{\max\{\sigma_\alpha, \min\{N, 1 + r_\alpha/r_{\min}\}\}}{\rho_\alpha},$$

once it becomes conforming. Parekh and Gallager proved that a cell departs a WFQ server at most one time slot later than it departs the underlying GPS server [6]. Consequently, FLB shaping, based on weighted fair queueing, bounds shaping delay to

$$\frac{\max\{\sigma_\alpha, \min\{N, 1 + r_\alpha/r_{\min}\}\}}{\rho_\alpha} + 1,$$

once a cell conforms to its connection's leaky-bucket parameters. Although rescheduling non-conforming cells can increase average cell shaping latency, the FLB shaper enforces tight bounds on *worst-case* delay and traffic distortions, independent of the conformance times of other connections' cells.

# 4 Self-Clocked Implementation

Rescheduling non-conforming cells simplifies the FLB shaper implementation without significantly distorting the leaky-bucket parameters of the resulting output streams. When a new cell arrives,

the shaper identifies the appropriate connection and computes the cell's conformance time; later, when the fair queueing server selects the cell, this conformance timestamp determines whether the shaper transmits the cell or reschedules it for later service. As a result, the FLB shaper ranks cells by their virtual finishing times in the fair queueing system, instead of sorting cells by conformance times. Although fair queueing implementations typically require a priority queue to sort cells by virtual finishing times [20], the FLB shaper implements self-clocked fair queueing using two sets of FIFO queues.

### 4.1 Per-Connection Queueing

Traffic shapers in high-speed networks require efficient hardware mechanisms for arbitrating amongst competing cells. A shaper can reduce sorting complexity by considering only the head-of-line cell from each active connection, while holding the remaining cells in per-connection queues. These queues can consist of logical FIFOs in a common shared memory, with an idle-address pool maintaining a list of available buffer slots. When the shaper ranks cells by conformance times, per-connection queueing reduces the number of cells and the range of conformance times that the sorting mechanism must handle. Similarly, per-connection FIFOs can reduce sorting complexity in the FLB shaper by limiting the range of virtual finishing times.

Under self-clocked fair queueing, if connection $\alpha$ is backlogged at time $t$, its head-of-line cell satisfies $S^\alpha \leq F^{serv} \leq F^\alpha$, since the cell has reached the head of its queue but has not completed service. Since $S^\alpha = F^\alpha - 1/r_\alpha$,

$$F^{serv} \leq F^\alpha \leq F^{serv} + 1/r_\alpha \quad \forall \alpha \in B(t).$$

Hence, at time $t$ all virtual finishing times fall in the range $[F^{serv}, F^{serv} + 1/r_{\min}]$, where $r_{\min} = \min_\omega\{r_\omega\}$ and $F^{serv}$ is the virtual finishing time of the cell currently in service. This *range* of virtual finishing times becomes a *finite set* under some practical restrictions on connection weights. If a SCFQ server has integer $1/r_\alpha$ values for all connections $\alpha$, then $F$ values are also integers; this property does not hold for WFQ servers, since $R(t)$ is a *continuous* function under weighted fair queueing. With integer virtual finishing times, a SCFQ server handles head-of-line cells with at most $1 + 1/r_{\min}$ different $F$ values, independent of the number of connections.

### 4.2 Sorting by Virtual Finishing Times

Under this finite set of virtual finishing times, a SCFQ implementation can consist of a small collection of FIFOs, with one queue for each $F$ value, as shown in Figure 4. The server arbitrates fairly amongst competing connections by sequencing through these FIFOs, draining one bin completely before proceeding to the next non-empty queue. The bin in service represents the current value of $F^{serv}$. When connection $\alpha$ has a new head-of-line cell, the server implicitly assigns a virtual

finishing time by placing the cell (or a pointer to the cell) in the queue $1/r_\alpha$ away from the FIFO currently in service. This ensures that connections receive service in proportion to their bandwidth requirements $r_\alpha \propto \rho_\alpha$.

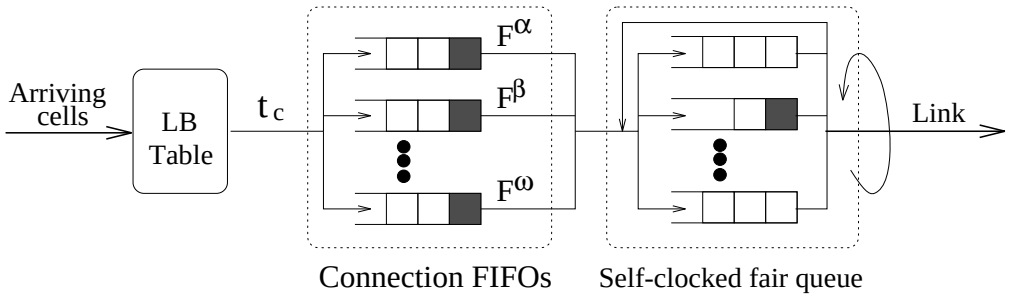



Figure 4: Self-clocked FLB architecture

In effect, the SCFQ implementation dynamically creates a fair schedule for the active connections, with each new head-of-line cell joining a FIFO based on its connection's weight; in this sense, the SCFQ implementation is similar to rate-based scheduling [16], although the SCFQ architecture is work-conserving and does not restrict the depth of the FIFO queues. For example, if all connections have equal weights $r_\alpha = 1$, then each head-of-line cell has a virtual finishing time of either $F^{serv}$ or $F^{serv} + 1$. Thus, the SCFQ server can simply toggle between two FIFO queues, with the link serving one FIFO, while new head-of-line cells join the other; actually, a *single* FIFO queue suffices, since all new head-of-line cells have virtual finishing time $F^{serv} + 1$ and can be served after any traffic already in the FIFO. In this case, we recognize SCFQ as a well known scheduling method that toggles between two waiting rooms, with the room in service closed to newcomers, who then enter the other room [22]. By ranking cells according to virtual finishing times, the FLB shaper avoids sorting across a wide range of cell conformance times; even with per-connection queueing, head-of-line cells can have arbitrary conformance times up to $1/\rho_{\min}$ into the future, requiring a large number of sorting FIFOs to distinguish between competing cells.

Since fair queueing defines connection bandwidths as *ratios* of weights, limiting these weights to the inverse of integers does not impose a significant restriction. The required number of FIFOs increases as the SCFQ server incorporates a wider range and granularity of connection weights. Suppose the server imposes $r_{\max}/r_{\min} = n$, with a granularity of $m$ connection weights between $r_{\min}$ and $r_{\max}$, by using weights $\{\frac{1}{m}, \frac{1}{2m}, \ldots, \frac{1}{nm}\}$; for example, if $n = 2$ and $m = 3$, the server could allow weights $\{1/2, 1/3, 1/4\}$. As a result, the SCFQ implementation requires at most $nm + 1$ FIFOs to sort fairly amongst the competing connections. This significantly reduces the implementation complexity of self-clocked fair queueing and FLB shaping, particularly for a small range of connection bandwidth requirements.

The FLB shaper can efficiently support a finer grain of connection rates through an *approxi-*

*mate* implementation of self-clocked fair queueing. In this approach, the shaper permits arbitrary connection weights $r_\alpha$ and associates each FIFO queue with a *range* of virtual finishing times; the server implicitly rounds each cell's $F$ value to select the appropriate FIFO in the calendar queue. Each connection $\alpha$ can maintain its full, non-integer virtual finishing time to correctly assign the $F^\alpha$ value for the next head-of-line cell. We are currently analyzing and evaluating the fairness properties of this approximate SCFQ scheme; other possible extensions address efficient support for a wider *range* of connection weights. These generalizations of the SCFQ implementation will be presented in a forthcoming paper.

By combining leaky-bucket control and a SCFQ server, the FLB shaper can perform traffic policing, traffic shaping, and fair queueing, as shown in Table 1. If non-conforming cells are dropped (or marked) on arrival, the FLB architecture implements traffic policing. For ingress policing, the aggregate rate of the input links does not exceed the transmission rate, so conforming cells proceed directly to the output link. With egress policing, however, the outgoing link transmits at a slower rate than the peak incoming traffic, so the SCFQ server plays a useful role in preserving connection leaky-bucket parameters. The FLB architecture can also implement fair traffic shaping by permitting non-conforming arrivals and transmitting only conforming cells. When rescheduling is disabled, the FLB architecture implements fair queueing without any restrictions on the arriving traffic streams.

| **Function** | **Admit non-conforming cells** | **Allow rescheduling in SCFQ** |
|---|---|---|
| Traffic policing | No | — |
| Traffic shaping | Yes | Yes |
| Fair queueing | Yes | No |

Table 1: Options in FLB implementation

# 5 Performance Evaluation

This section evaluates the FLB shaper's effectiveness at limiting interference between ATM connections at network ingress and egress points. The simulation experiments compare the FLB architecture with a shaper that transmits conforming cells in order of their conformance times, breaking ties in FIFO order. The two schemes are compared based on the burstiness $\sigma^{out}$ of their resulting output streams; if a connection requests shaping with leaky-bucket parameters $(\sigma, \rho)$, then $\sigma^{out}$ is the smallest value such that the resulting output stream is $(\sigma^{out}, \rho)$-compliant.

## 5.1 Ingress Shaping

The ingress simulation model consists of a single input link that operates at the same rate as the output port, with most one cell arriving in each time slot. To evaluate the two shaping schemes on a challenging traffic pattern, the ingress experiments consider a scenario similar to Figure 2. The experiment generates a large number of cells with conformance time $T$ by having each connection $i$, with $\sigma_i = 1$, $\rho_i = 1/(T - 2i)$, submit two cells at $2i$ and $2i + 1$, where $i = 1, 2, \ldots, N - 1$. Although connection $i$'s first cell is conforming on arrival, the second cell does not become conforming until time $T$. As soon as these low-bandwidth connections finish submitting their two-cell packets, a high-rate connection begins periodically transmitting conforming cells, as shown in Figure 5.

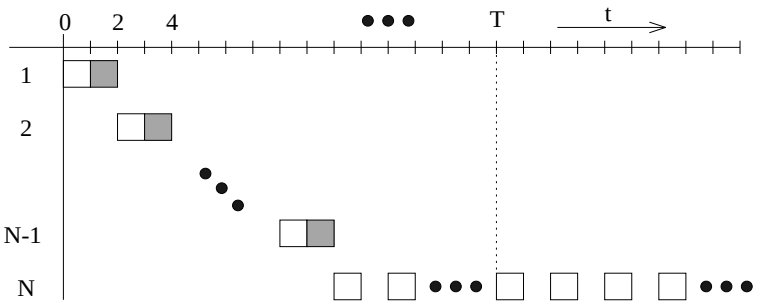
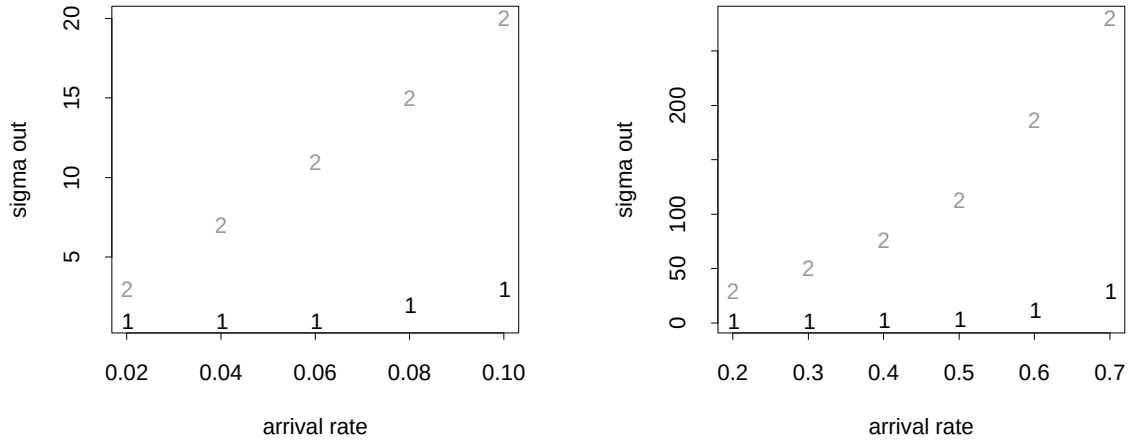



Figure 5: Ingress traffic arrival pattern

Figure 6(a) shows $\sigma^{out}$ for the high-bandwidth connection with leaky-bucket parameters $(1, 0.109)$, with 200 low-rate connections and $T = 480$. FIFO scheduling of conforming cells significantly distorts the connection leaky-bucket parameters, particularly when the high-bandwidth connection submits cells at a high rate, close to its shaping rate $\rho$. This occurs because, starting at time $T$, the FIFO shaper consumes 200 consecutive time slots serving the newly-conforming cells from the low-bandwidth connections, allowing the high-rate connection to accumulate a large burst of backlogged traffic. Even though the high-bandwidth connection complies with its leaky-bucket parameters at the input port, the shaper significantly distorts the resulting output stream. In contrast, FLB shaping closely preserves connection traffic descriptors by interleaving the high-bandwidth traffic with the newly-conforming cells from the low-rate connections. The low-bandwidth connections do not experience any traffic distortions under either shaper model.

The next experiment considers a similar traffic pattern, with 1000 low-rate connections and $T = 10,000$. While the high-rate connection has leaky-bucket parameters $(1, 0.88)$, the experiment varies the connection's cell arrival rate, as shown in Figure 6(b). As in Figure 6(a), FIFO scheduling permits low-rate connections to deny service to the conforming cells from the high-bandwidth connection. This effect becomes more pronounced as the high-bandwidth connection submits cells at a higher rate, closer to its average sustainable rate. However, in this experiment, the wider range




(a) With 200 low-bandwidth connections (b) With 1000 low-bandwidth connections

Figure 6: Performance of high-bandwidth connection under ingress shaping ("1" indicates results for the FLB shaper, while "2" denotes the FIFO shaper model).

of connection $\rho$ values causes both shaping schemes to distort the high-bandwidth connection's $\sigma^{out}$ parameter, although the FLB shaper introduces significantly less distortion. To accommodate a broader spectrum of $\rho$ values, the FLB architecture can group connections with similar bandwidth requirements and arbitrate fairly amongst the competing groups.

## 5.2 Egress Shaping

To evaluate traffic shaping at the network egress, the simulation model has an input port that operates $\tau$ times faster than the output link, although the aggregate mean rate of the connections does not exceed the speed of the output port. In general, demultiplexing to lower-speed links amplifies the effects of traffic collisions, since transient input load can generate a large backlog of conforming cells in the shaper buffer. This experiment consists of 10 compliant connections with shaping rates $\rho = 0.05$; each incoming stream consists of a 20-cell burst, at the rate of the input link, with one connection's burst following another. At the same time, a high-rate connection has periodic cell arrivals at rate 0.4, with leaky-bucket parameters $(1, 0.5)$.

As $\tau$ increases, this high-rate connection must endure more interference from the bursty low-rate connections, as shown in Figure 7. Fairness plays a crucial role in this context. The network can shape a connection's traffic at the network entry point, then closely preserve the leaky-bucket descriptors throughout the connection's route, but then ultimately disrupt these traffic parameters

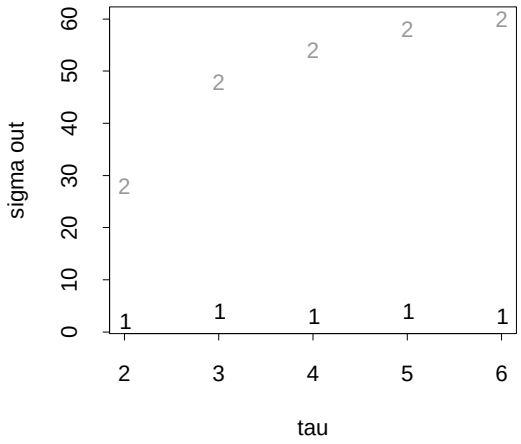



Figure 7: Performance of high-bandwidth connection under egress shaping ("1" indicates results for the FLB shaper, while "2" denotes the FIFO shaper model).

at the egress port. This port may feed an end host or even another subnetwork with lower speed links. Fair shaping at the egress port can accommodate variations in traffic flow and prepare the connection for the end host or the next subnetwork in the route.

# 6 Conclusion

Traffic shaping and fair queueing both regulate the interaction between competing connections in an ATM network. While shaping enforces connection traffic descriptors, fair queueing ensures that each connection receives its share of the link bandwidth on a fine time-scale. Combining leaky-bucket shaping and fair queueing produces a shaper architecture that preserves connection traffic descriptors and facilitates an efficient implementation. The FLB shaper limits the pernicious effects of collisions between competing cells.

Ultimately, fair queueing algorithms require compromises between implementation complexity and the accuracy in approximating generalized processor sharing. As an extension of the results presented here, we are studying architectures that guarantee efficient shaping of a broader spectrum of connection weights, facilitate more diverse link-sharing policies in ATM networks, and use approximate implementations of self-clocked fair queueing to support a finer granularity of connection bandwidth requirements without increasing hardware complexity.

In addition to investigating efficient implementations of fair queueing, we are considering optimizations for rescheduling non-conforming cells in the FLB shaper. The shaper can improve performance by considering alternate cells for transmission when a non-conforming cell has the

smallest virtual finishing time. With a moderate increase in server speed, the shaper can reduce the likelihood of generating idle transmission slots when conforming cells await service. In addition, if some connections do not request traffic shaping, the server can fill output slots with cells from these connections. The combination of leaky-bucket control, fair queueing, and rescheduling results in an effective and efficient shaper implementation.

# Appendix A Throughput Discrepancy Bound

**Theorem 3** *For all times $t$ and connections $\alpha$,*

$$W_\alpha^{WFQ}(0,t) - W_\alpha^{GPS}(0,t) \leq \min\{N-1, r_\alpha/r_{\min}\} L_{\max},$$

*where $N$ is the number of connections, $r_{\min} = \min_\omega\{r_\omega\}$, and $L_{\max}$ is the maximum packet (cell) length.*

**Proof.** Since WFQ and GPS are both work-conserving, they provide the same total service over any time interval. That is,

$$\sum_\omega W_\omega^{GPS}(0,t) = \sum_\omega W_\omega^{WFQ}(0,t)$$

for all $t$. GPS cannot lag far behind WFQ for any connection $\alpha$, since WFQ lags at most $L_{\max}$ behind GPS for *each* of the $N$ connections. Since $W_\omega^{GPS}(0,t) \leq W_\omega^{WFQ}(0,t) + L_{\max}$ for all $N$ connections $\omega$ [6], $W_\alpha^{WFQ}(0,t) - W_\alpha^{GPS}(0,t) \leq (N-1)L_{\max}$.

To prove the $r_\alpha/r_{\min}$ portion of the throughput bound, we consider the transmission of cell $k$ of connection $\alpha$ that enters service at time $t_1$ under WFQ. The difference $W_\alpha^{WFQ}(0,t) - W_\alpha^{GPS}(0,t)$ reaches its largest value when some connection $\alpha$ cell completes service under WFQ. Hence, it suffices to prove the throughput discrepancy bound at time $t_2$, when WFQ finishes transmitting the cell. Two possibilities emerge at time $t_1$. Either WFQ is already ahead of GPS in serving connection $\alpha$ or GPS does not yet lag behind WFQ. If GPS does not lag behind WFQ at $t_1$, then $W_\alpha^{WFQ}(0,t_1) \leq W_\alpha^{GPS}(0,t_1)$. This implies:

$$\begin{aligned} W_\alpha^{WFQ}(0,t_2) &= W_\alpha^{WFQ}(0,t_1) + L_\alpha^k \\ &\leq W_\alpha^{GPS}(0,t_1) + L_\alpha^k \\ &\leq W_\alpha^{GPS}(0,t_2) + L_\alpha^k \\ &\leq W_\alpha^{GPS}(0,t_2) + L_{\max}(r_\alpha/r_{\min}) \end{aligned}$$

The equality holds because WFQ serves cell $k$ of connection $\alpha$ for the entire interval $(t_1,t_2)$. The first inequality holds because $W_\alpha^{WFQ}(0,t_1) \leq W_\alpha^{GPS}(0,t_1)$, while the second inequality is a consequence

of $t_1 < t_2$. Since $L_{\max} \geq L_\alpha^k$ and $r_\alpha \geq r_{\min}$, the discrepancy bound holds when $W_\alpha^{WFQ}(0,t_1) \leq W_\alpha^{GPS}(0,t_1)$.

Suppose, instead, that $W_\alpha^{WFQ}(0,t_1) > W_\alpha^{GPS}(0,t_1)$. Since both GPS and WFQ are work conserving, WFQ must lag behind GPS for *at least one* connection at time $t_1$. Let cell $j$ be the head cell of this connection $\beta$ at time $t_1$ under WFQ service. Since WFQ lags behind GPS in serving connection $\beta$, $S_j^\beta < R(t_1)$; similarly, since GPS lags WFQ for connection $\alpha$, $S_k^\alpha > R(t_1)$. However, WFQ still selects connection $\alpha$ for service at time $t_1$, instead of choosing connection $\beta$, so $F_k^\alpha \leq F_j^\beta$. When WFQ finishes serving cell $k$ of connection $\alpha$ at time $t_2$, the underlying GPS server still has $(F_k^\alpha - R(t_2))$ virtual time remaining before this cell completes service. So, GPS must perform $r_\alpha(F_k^\alpha - R(t_2))$ additional work for connection $\alpha$ to complete the transmission of cell $k$. Thus,

$$\begin{aligned} W_\alpha^{WFQ}(0,t_2) - W_\alpha^{GPS}(0,t_2) &= r_\alpha(F_k^\alpha - R(t_2)) \\ &\leq r_\alpha(F_j^\beta - R(t_2)) \\ &\leq r_\alpha(F_j^\beta - S_j^\beta) \\ &= r_\alpha(L_j^\beta / r_\beta) \\ &\leq r_\alpha(L_{\max}/r_{\min}) \end{aligned}$$

The first inequality holds because $F_j^\beta \geq F_k^\alpha$, while the second inequality stems from $S_j^\beta \leq R(t_1) < R(t_2)$; the identity $F_j^\beta = S_j^\beta + L_j^\beta / r_\beta$ simplifies the expression. Since $L_{\max} \geq L_j^\beta$ and $r_\beta \geq r_{\min}$, the discrepancy bound holds at time $t_2$ when $W_\alpha^{WFQ}(0,t_1) > W_\alpha^{GPS}(0,t_1)$. Thus, the bound holds for all $t$. □

Previously proven for the special case of equal weights $r_\alpha$ [21], this theorem complements the relation $W_\alpha^{GPS}(0,t) - W_\alpha^{WFQ}(0,t) \leq L_{\max}$ [6]. It can be shown that both of these throughput discrepancy bounds are tight.

## References


[1] J. Turner, "New directions in communications, or which way to the information age?," *IEEE Communication Magazine*, vol. 24, pp. 8–15, Oct. 1986.

[2] I. Cidon and I. S. Gopal, "PARIS: An approach to integrated high-speed private networks," *Intl. Journal of Digital and Analog Cabled Systems*, vol. 1, pp. 77–86, Apr. 1988.

[3] M. Sidi, W.-Z. Liu, I. Cidon, and I. Gopal, "Congestion control through input-rate regulation," *IEEE Trans. Communications*, vol. 41, pp. 471–477, Mar. 1993.

[4] R. L. Cruz, "A calculus for network delay, part I: Network elements in isolation," *IEEE Trans. Information Theory*, vol. 37, pp. 114–131, Jan. 1991.

[5] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorthm," *Proc. of ACM SIGCOMM*, pp. 3–12, 1989. (Also *J. Internetworking: Research and Experience*, pp. 3–26, September 1990).

[6] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks – the single node case," in *Proc. IEEE INFOCOM*, pp. 915–924, 1992. (Also *IEEE/ACM Trans. Networking*, pp. 344–357, June 1993).

[7] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks – the multiple node case," in *Proc. IEEE INFOCOM*, pp. 521–530, 1993. (Also *IEEE/ACM Trans. Networking*, pp. 137–150, April 1994).

[8] J. R. Davin and A. T. Heybey, "A simulation study of fair queueing and policy enforcement," *Computer Communication Review*, pp. 23–29, Oct. 1990.

[9] S. J. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *Proc. IEEE INFOCOM*, pp. 636–646, 1994.

[10] H. J. Chao and N. Uzun, "A VLSI sequencer chip for ATM traffic shaper and queue manager," *IEEE J. of Solid-State Circuits*, vol. 27, pp. 1634–1643, Nov. 1992.

[11] H. J. Chao, "Design of leaky bucket access control schemes in ATM networks," in *Proc. International Conference on Communications*, pp. 180–187, June 1991.

[12] P. E. Boyer, F. M. Guillemin, M. J. Servel, and J.-P. Coudreuse, "Spacing cells protects and enhances utilization of ATM network links," *IEEE Network Magazine*, pp. 38–49, Sept. 1992.

[13] K. van der Wal, M. Dirksen, and D. Brandt, "Implementation of a police criterion calculator based on the leaky bucket algorithm," in *Proc. GLOBECOM*, pp. 713–718, November/December 1993.

[14] E. Wallmeier and T. Worster, "A cell spacing and policing device for multiple virtual connections on one ATM pipe," in *Proc. RACE Workshop on Network Planning and Evaluation*, Apr. 1991.

[15] H. Zhang and D. Ferrari, "Rate-controlled static-priority queueing," in *Proc. IEEE INFOCOM*, pp. 227–236, June 1993.

[16] C. R. Kalmanek, H. Kanakia, and S. Keshav, "Rate controlled servers for very high-speed networks," in *Proc. GLOBECOM*, Dec. 1990.

[17] H. Zhang and S. Keshav, "Comparison of rate-based service disciplines," in *Proc. of ACM SIGCOMM*, pp. 113–121, Sept. 1991.

[18] L. Zhang, "Virtual Clock: A new traffic control algorithm for packet-switched networks," *ACM Trans. Computer Systems*, vol. 9, no. 2, pp. 101–124, 1991.

[19] S. J. Golestani, "Congestion-free communication in high-speed packet networks," *IEEE Trans. Communications*, vol. 39, pp. 1802–1812, Dec. 1991.

[20] S. Keshav, "On the efficient implementation of fair queueing," *J. Internetworking: Research and Experience*, vol. 2, pp. 157–173, Sept. 1991.

[21] A. G. Greenberg and N. Madras, "How fair is fair queueing?," *J. of the ACM*, vol. 39, pp. 568–598, July 1992.

[22] A. Fraser, "Group contention on a demand-shared bus," Feb. 1977. Bell Laboratories internal memorandum.