Hardware-Efficient Fair Queueing Architectures for High-Speed Networks

Jennifer L. Rexford*
Department of EECS
University of Michigan
Ann Arbor, MI 48109

Albert G. Greenberg
Network Services Research
AT&T Bell Laboratories
Murray Hill, NJ 07974

Flavio G. Bonomi
AT&T Platform Organization
AT&T Network Systems
Red Bank, NJ 07701

Abstract

In emerging communication networks, a single link may carry traffic for thousands of connections with different traffic parameters and quality-of-service requirements. High-speed links, coupled with small packet/cell sizes, require efficient switch architectures that can handle cell arrivals and departures every few microseconds, or faster. This paper presents a collection of self-clocked fair queueing (SCFQ) architectures amenable to efficient hardware implementation in network switches. Exact and approximate implementations of SCFQ efficiently handle a moderate range of connection bandwidth parameters, while hierarchical arbitration schemes scale to a large range of throughput requirements. Simulation experiments demonstrate that these architectures divide link bandwidth fairly on a small time scale, preserving connection bandwidth and burstiness properties.

1 Introduction

Emerging broadband integrated service digital networks, based on asynchronous transfer mode (ATM) technology, carry 53-byte cells from various connections with different traffic characteristics and performance requirements. Admission control policies govern whether the network can accommodate a new connection, based on its traffic descriptors and quality-of-service parameters, while cell-scheduling algorithms govern the interaction between connections at each network link. Although complex cell-scheduling algorithms can significantly improve connection admissibility and performance, high-speed links require simple and efficient arbitration schemes.

To minimize implementation complexity, many existing ATM switches transmit cells in first-in first-out order. However, FIFO scheduling does not insulate cells from heavy traffic or malicious behavior in other connections [1]. An unexpected burst of cells can significantly delay and distort other traffic contending for the link, violating the expectations of downstream switches in the network; this, in turn, can increase end-to-end delay, jitter, and cell loss rates. Instead, network switches should arbitrate fairly amongst competing connections, or groups of connections, to provide delay and throughput guarantees.

Fair scheduling algorithms guarantee performance by providing a minimum service rate to each busy, or *backlogged*, connection on a relatively small time scale. Although fair queueing algorithms have been shown to have excellent performance properties [2–12], the complexity of the algorithms inhibits their use in high-speed switches, which must select a cell for transmission every few microseconds, if not faster. In addition, performance must scale to tens of thousands of connections, multiplexed onto a single link, with connection throughput requirements varying over a wide range. To overcome these problems, this paper presents a collection of SCFQ architectures conducive to efficient hardware implementation in high-speed network switches. The architectures are event-driven, manipulating a small, bounded number of cells or connections on each cell arrival and departure.

Section 2 describes how fair queueing algorithms assign logical timestamps to arriving cells; a practical fair queueing architecture requires a simple and efficient sorting mechanism to locate the cell with the smallest key. Capitalizing on some unique features of SCFQ, Section 3 proposes exact and approximate sorting schemes that efficiently handle connections with a moderate range of throughput requirements. Section 4 improves the implementation and performance properties of SCFQ by grouping connections with similar throughput parameters. This hierarchical approach arbitrates fairly between a small number of groups with diverse rate requirements, with each group arbitrating between a large number of connections with

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

similar bandwidth parameters. As shown by the simulation experiments in Section 5, hierarchical arbitration schemes can outperform an exact SCFQ server when the link handles a large number of connections with diverse bandwidth requirements. The paper concludes in Section 6 with a discussion of future research directions.

These architectures complement previous work on fair queueing algorithms and analytical studies of their fairness properties [2–12]; this paper extends these works by introducing efficient implementations that retain or improve the basic performance characteristics. The architectures in Section 3 relate to other mechanisms for effective rate control in high-speed networks [13–16] and recent efforts to implement high-speed packet/cell sorting in hardware [17–19]. The arbitration schemes in Section 4 relate to ongoing work to define a hierarchical framework for link sharing in integrated packet networks [20, 21].

2 Fair Cell Scheduling

A fair queueing algorithm multiplexes connections onto a single link that transmits one cell in each time slot; for stability, the link rate should exceed the sum of the connections' sustainable cell rates.

2.1 Fairness Paradigm

A switch could guarantee minimum service rates by dedicating a private link to each virtual connection, instead of multiplexing traffic onto a single shared resource. Although this is not feasible in practice, it provides a useful paradigm for governing the interaction between competing connections. *Generalized processor sharing* (GPS) models this abstraction by *continuously* dividing link bandwidth among the backlogged connections, in proportion to their weights r_α [2–4]. Hence, a backlogged connection α transmits traffic continuously at rate $r_\alpha R'(t)$, where

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

$B(t)$ is the set of backlogged connections at time t. The GPS server's progress is tracked by the virtual clock $R(t)$, with $R(0) = 0$.

If cell k from connection α arrives at time t, it completes service at virtual finishing time F_α^k, where

$$F_\alpha^k = \max\{F_\alpha^{k-1}, R(t)\} + 1/r_\alpha ,$$

independent of future arrivals. GPS begins serving the cell when it arrives to an empty connection queue (at virtual time $R(t)$) or when the connection's previous cell departs (at virtual time F_α^{k-1}). The $1/r_\alpha$ term represents the normalized work required to service the new cell, assuming a fixed cell size, as in ATM. By guaranteeing a minimum throughput to each connection, a GPS server can preserve each connection's throughput and burstiness parameters [3, 4].

2.2 Packetized Algorithms

GPS cannot be implemented, since it requires preemption of the link resource on an arbitrarily small time scale. Weighted fair queueing (WFQ) approximates this idealized model by ranking cells by the time they would complete service under GPS, in the absence of future arrivals [2]. In each time slot, a WFQ server transmits the cell with the smallest F value, among the cells already queued for service (breaking ties arbitrarily). This approach closely tracks the underlying GPS reference model in terms of both throughput and delay. WFQ never lags more than one cell behind GPS in servicing a connection; similarly, a cell never completes service more than one time slot later than it would under GPS [3].

Although WFQ approximates GPS on a small time scale, computing $R(t)$ is complex, since it can require "simulating" multiple GPS service events during a single cell transmission slot [5, 6]. Self-clocked fair queueing (SCFQ) [7–10] avoids the burden of computing $R(t)$ by using the virtual finishing time F^{serv} of the cell *currently in service* as an estimate of current virtual time. That is,

$$F_\alpha^k = \max\{F_\alpha^{k-1}, F^{serv}\} + 1/r_\alpha .$$

Analysis [8, 9] and simulation [7] indicate that SCFQ servers can provide throughput and delay guarantees; however, these performance bounds are weaker than the equivalent WFQ results, particularly when the server handles a large number of connections with a wide range of bandwidth requirements [9].

To simplify implementation, a SCFQ server can postpone computing a cell's F value until the cell reaches the head of its connection queue [10]. Cell k has a virtual finishing time of $F^{serv} + 1/r_\alpha$ when it reaches the head of its connection queue, whether the cell arrives to an idle connection or reaches the head-of-line position when cell $k-1$ departs; this is not true for WFQ servers, since virtual finishing times depend on the value of $R(t)$ *at cell arrival*. Delaying the computation of virtual finishing times allows a SCFQ server to maintain a single F_α value for each connection α, instead of timestamping individual cells.

3 Self-Clocked Architecture

Implementations of fair queueing algorithms require a sorting mechanism to rank cells by their virtual finishing times. This section presents an efficient

SCFQ architecture that schedules cells using two sets of FIFO queues, instead of a complex priority queue.

3.1 Architecture

Since virtual finishing times are monotonically increasing within each connection, only head-of-line cells need to compete for link access [5, 10]. These head-of-line cells have a limited range of virtual finishing times. Since F^{serv} increases with time, a backlogged connection α must satisfy $F_\alpha \leq F^{serv} + 1/r_\alpha$. In addition, $F_\alpha \geq F^{serv}$, since the connection's head cell has not been selected for service yet. As a result,

$$F_\alpha \in [F^{serv}, F^{serv} + 1/r_{\min}] \quad \forall \alpha \in B(t),$$

where $r_{\min} = \min_\omega\{r_\omega\}$ and $B(t)$ is the set of backlogged connections at time t. The bounded range of connection F values permits the server to compute virtual finishing times using modulo arithmetic, using any base that exceeds $1/r_{\min}$.

Moreover, this bounded range of virtual finishing times becomes a *finite set* under some practical restrictions on connection weights. In particular, if a SCFQ server has integer $1/r_\alpha$ values for all connections α, then F values are also integers. This property does not hold for WFQ servers, since $R(t)$ is a *continuous* function. With integer virtual finishing times, a SCFQ server arbitrates among head-of-line cells with at most $1 + 1/r_{\min}$ different F values, independent of the number of connections. Limiting connection weights to the inverse of integers does not impose a significant restriction, since fair queueing assigns bandwidth based on the *ratios* of connection weights.

Under this finite set of virtual finishing times, a SCFQ implementation can consist of per-connection FIFOs and a small set of FIFO sorting bins, as shown in Figure 1. Each of the $1 + 1/r_{\min}$ sorting bins corresponds to a single F value; the bin in service represents the current value of F^{serv}. The server arbitrates fairly among competing connections by cycling through these bins, transmitting all cells from one bin before proceeding to the next non-empty bin. When connection α 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; a cell enters a sorting bin upon arrival to an empty connection or upon the departure of the connection's previous cell. This dynamically creates a fair schedule for the backlogged connections, based on their rate parameters r_α, without explicitly storing any F values.

3.2 Implementation Cost

The per-connection FIFO queues can consist of linked lists in a common shared memory, with an idle-

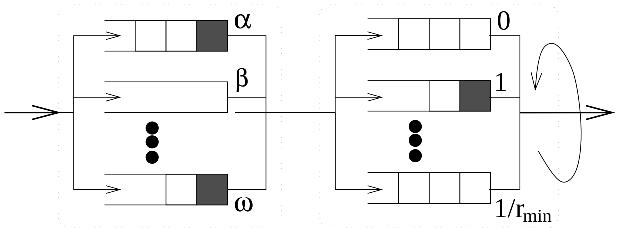

Figure 1: SCFQ architecture with integer $1/r_\alpha$ values

address pool maintaining a list of available buffer slots. The switch stores an arriving cell in an idle memory location and appends the connection's linked list; upon cell departure, the switch returns the free memory address to the idle-address pool. The sorting bins can have a similar structure, storing cell pointers or connection identifiers to select the appropriate head-of-line cell for transmission. Each cell contains a connection identifier, while a table stores the integer $1/r_\alpha$ and the head/tail pointers for each connection queue.

Each cell arrival and departure triggers at most one $F^{serv} + 1/r_\alpha$ computation to determine the appropriate sorting bin for a new head-of-line cell. This requires at most one *mod*-$(1 + 1/r_{\min})$ adder for each link, instead of expensive per-connection or per-cell logic; the arriving/departing cell uniquely identifies which connection α to process. The required number of bins increases as the SCFQ server incorporates a wider range and granularity of connection weights. The architecture can service connections with weights $r_\alpha \in \{\frac{1}{1}, \frac{1}{2}, \ldots \frac{1}{b-1}\}$ using only b bins. Finally, the sorting mechanism requires logic to transmit a cell and locate the next non-empty sorting bin. This significantly reduces the implementation complexity of self-clocked fair queueing, particularly for a small range of connection bandwidth parameters.

The architecture can support a finer grain of connection rates, without increasing the number of bins, through an *approximate* version of SCFQ. In this approach, the shaper permits arbitrary connection weights $r_\alpha \geq r_{\min}$ and associates each bin with a *range* of virtual finishing times to reduce the number of sorting bins; the server implicitly rounds each cell's F value to select the appropriate FIFO in the calendar queue. In this scheme, a new head-of-line cell in connection α joins the sorting bin either $\lfloor 1/r_\alpha \rfloor$ or $\lceil 1/r_\alpha \rceil$ away from the current bin F^{serv}. This requires each connection to store its exact weight $1/r_\alpha$, as well as a fractional portion f_α from the last head-of-line cell. When a new cell ascends to the head-of-line position, the server implicitly assigns a virtual finishing time of $F^{serv} + \lfloor 1/r_\alpha + f_\alpha \rfloor$ and updates the remainder

$$f_\alpha = (1/r_\alpha + f_\alpha) - \lfloor 1/r_\alpha + f_\alpha \rfloor.$$

(A similar computation is used in a recent frame-based scheduling algorithm that allows connections to request a non-integer number of slots per frame [16]; each connection receives the integer part of its slot assignment in each frame and can contend for additional slots based on the accumulated fractional portion. In contrast, the SCFQ architecture is work-conserving and avoids per-connection logic.)

4 Hierarchical Fair Queueing

Although the architectures in Section 3 efficiently support a a moderate range of connection rates, handling a wide range of bandwidth parameters would require a large number of sorting bins and the associated sequencing logic. A more scalable approach groups connections based on their throughput parameters. This results in a two-level scheduler, where the first stage handles a large number of connections with similar rates (e.g., rates within a small multiple of each other), while the second stage services a small number of groups with diverse bandwidth requirements. Weights ϕ_i coordinate bandwidth sharing between competing groups.

4.1 Static Group Weights

Thus, each group consists of a fair queueing server that arbitrates among its connections, as described in Section 3, while a separate server determines which group has access to the outgoing link. Since the system has a small number of groups, the second-stage arbiter can efficiently arbitrate among the competing groups; for example, the server could employ a small tree of comparators to identify the backlogged group i with the minimum virtual finishing time F_i. Figure 2 summarizes the operation of the hierarchical SCFQ architecture, where group weights ϕ_i are fixed for each group i. Hence, only an idle group shares its excess bandwidth with other competing groups. This hierarchical link-sharing model is particularly useful when groups correspond to different institutions, service classes, or protocol families [20, 21].

The second-stage SCFQ arbiter ensures that group i can receive at least $\phi_i / \sum_j \phi_j$ of the link bandwidth; in turn, connection α_i in group i can receive a minimum throughput of

$$\frac{r_{\alpha_i}}{\sum_{\omega_i} r_{\omega_i}} \cdot \frac{\phi_i}{\sum_j \phi_j},$$

independent of the traffic in other connections; this suggests choosing group weights ϕ_i that are proportional to the sum of the connection weights r_{α_i} to achieve an appropriate minimum bandwidth at the connection level. The two-stage architecture performs a small, bounded number of operations on each cell arrival and departure, as shown in Figure 2. A cell arrival can trigger the computation of the connection's virtual finishing time F_{α_i}, as well as the group's virtual finishing time F_i in the second-stage arbiter.

In each transmission slot, the second-stage server selects the backlogged group i with the smallest virtual finishing time F_i. This group then selects the backlogged connection α_i with the smallest F_{α_i}, as described in Section 3. A backlogged group does not sequence through its sorting bins unless it is selected for service; this ensures that each connection receives its fair share of its group's bandwidth allocation, on a relatively small time scale. Recall that the FIFO sorting bins *implicitly* represent the values of F_i^{serv} and F_{α_i} in each group i, while F_i and F^{serv} are *explicitly* stored in the second-stage SCFQ arbiter; this server handles a bounded range of group F values, permitting modulo arithmetic and registers that store values ranging from 0 to $1/\min_i\{\phi_i\}$.

4.2 Dynamic Group Weights

Sharing excess bandwidth fairly at the *connection* level requires the group weights ϕ_i to reflect the bandwidth requirements of the *backlogged* connections in each group i. This suggests a heuristic algorithm that dynamically adjusts group weights ϕ_i and virtual finishing times F_i as connections become backlogged or empty. For example, a group could assign

$$\phi_i(t) = \sum_{\alpha_i \in B_i(t)} r_{\alpha_i}$$

where $B_i(t)$ is the set of backlogged connections in group i at time t. Changes to $B_i(t)$ occur upon cell arrivals and departures, facilitating efficient updates to ϕ_i by adding (subtracting) r_{α_i} when connection α_i becomes backlogged (empty).

Changes in ϕ_i affect the virtual finishing time F_i for group i. When a cell departure decreases ϕ_i, the new group weight can directly affect the new value of F_i. However, F_i should also change when a cell arrives to an empty connection, even if the *group* is already backlogged. The server could postpone recomputing F_i until the next cell departure from group i, but this could significantly delay service to the group's connections, especially if multiple connections become backlogged in the interim. Instead, the server can decrease

Cell Arrival

cell arrives for connection α_i in group i;
if (connection α_i is empty)
    $F_{\alpha_i} = F_i^{serv} + 1/r_{\alpha_i}$;
if (group i is empty)
    $F_i = F^{serv} + 1/\phi_i$;
append cell to connection α_i in group i;

Cell Departure

find backlogged group i with min F_i;
find backlogged connection α_i with min F_{α_i};
transmit head cell from connection α_i;
$F^{serv} = F_i$; $F_i^{serv} = F_{\alpha_i}$;
if (connection α_i is still backlogged)
    $F_{\alpha_i} = F_i^{serv} + 1/r_{\alpha_i}$;
if (group i is still backlogged)
    $F_i = F^{serv} + 1/\phi_i$;

Figure 2: Hierarchical SCFQ algorithm with static group weights

F_i to reflect the new value of ϕ_i, as long as this does not overly penalize the other backlogged groups.

The algorithm for dynamic weights adjusts F_i when a cell arrives to an empty connection α_i, even if the group is already backlogged, as shown in Figure 3. Each group maintains the current value of ϕ_i as well as ϕ_i^{old}, the value of ϕ_i the last time F_i was assigned; ϕ_i^{old} is set to ϕ_i whenever the algorithm assigns a new value to F_i. When ϕ_i becomes sufficiently larger than ϕ_i^{old}, due to one or more newly backlogged connections, the algorithm decreases the value of F_i to serve group i sooner. The server never decreases F_i below F^{serv}; otherwise, group i would unnecessarily penalize the other backlogged groups by claiming past bandwidth.

The algorithm first tries to recompute F_i based on the previous departure from the group, at virtual time $F_i - 1/\phi_i^{old}$, as long as the new F_i value would not be smaller than F^{serv}. If the existing value of F_i is too close to F^{serv}, the algorithm then computes a virtual finishing time as if the group had just received service, resulting in $F^{serv} + 1/\phi_i$. The group settles for this new virtual finishing time, unless it exceeds the previous value of F_i. This approach reduces F_i to respond quickly to newly backlogged connections without unduly delaying service to other groups.

5 Evaluation

A collection of simulation experiments focus on the unique aspects of each self-clocked fair queueing architecture. The results show that approximate sorting, with calendar queues, performs well for a moderate range of connection rate parameters. For more diverse throughput requirements, a hierarchical arbiter with dynamic weights closely preserves connection traffic descriptors, sometimes outperforming an exact implementation of self-clocked fair queueing.

5.1 Performance Metrics

The simulation experiments focus on how the SCFQ architectures affect connection throughput and burstiness parameters. Although the architectures do not require connections to obey any particular traffic specification, the simulations evaluate traffic distortion by comparing the *leaky-bucket* parameters of the incoming and outgoing cell streams. Leaky-bucket parameters are widely used to characterize a connection's average bandwidth and short-term burstiness. A leaky-bucket characterization constrains a connection α by generating tokens at rate ρ_α, where the token bucket holds at most σ_α tokens; an arriving cell must claim a token before entering the system [22, 23].

Multiplexing a connection with other network traffic results in an outgoing stream that is $(\sigma_\alpha^{out}, \rho_\alpha)$-compliant, where σ_α^{out} may exceed σ_α; fair queueing schemes can bound this distortion as a connection's cells travel through the network. For example, a GPS server with weights $r_\alpha \propto \rho_\alpha$, and $\sum_\omega \rho_\omega < 1$ can continuously serve connection α cells at rate ρ_α, or faster; this ensures that GPS preserves the leaky-bucket parameters for each connection (i.e., $\sigma_\alpha^{out} = \sigma_\alpha$) [3, 4]. The simulator also measures the *average* traffic distortion by feeding a connection's outgoing cells to a $(\sigma_\alpha, \rho_\alpha)$ traffic shaper and accumulating the mean shaping delay $\overline{\sigma}_\alpha^{out}$, in units of leaky-bucket tokens. Ideally, all cells are leaky-bucket conforming when they leave the SCFQ server, resulting in zero shaping delay ($\overline{\sigma}_\alpha^{out} = 0$).

5.2 Sorting Mechanisms

This subsection considers how the exact and approximate sorting schemes affect connections' traffic characteristics. The traffic consists of 10 connections with weight 1/3, 20 connections with weight 1/24, and 100 connections with weight 1/40. An exact implementation of SCFQ requires 41 ($= 1 + 1/\frac{1}{40}$) sorting bins; the simulation experiments compare this scheme to an approximate architecture that employs only 7 sorting FIFOs, where each bin corresponds to a range of 7 virtual finishing times. The first experiment demonstrates how the architectures divide bandwidth fairly among backlogged connections on a relatively

Cell Arrival

cell arrives for connection α_i in group i;
if (connection α_i is empty)
  $F_{\alpha_i} = F_i^{serv} + 1/r_{\alpha_i}$;
  $\phi_i = \phi_i + r_{\alpha_i}$;
  if (group i is empty)
    $F_i = F^{serv} + 1/\phi_i$; $\phi_i^{old} = \phi_i$;
  else if ($F^{serv} \leq F_i - 1/\phi_i^{old} + 1/\phi_i$)
    $F_i = F_i - 1/\phi_i^{old} + 1/\phi_i$; $\phi_i^{old} = \phi_i$;
  else if ($F_i > F^{serv} + 1/\phi_i$)
    $F_i = F^{serv} + 1/\phi_i$; $\phi_i^{old} = \phi_i$;
append cell to connection α_i in group i;

Cell Departure

find backlogged group i with min F_i;
find backlogged connection α_i with min F_{α_i};
transmit head cell from connection α_i;
$F^{serv} = F_i$; $F_i^{serv} = F_{\alpha_i}$;
if (connection α_i is still backlogged)
  $F_{\alpha_i} = F_i^{serv} + 1/r_{\alpha_i}$;
else
  $\phi_i = \phi_i - r_{\alpha_i}$;
if (group i is still backlogged)
  $F_i = F^{serv} + 1/\phi_i$; $\phi_i^{old} = \phi_i$;

Figure 3: Hierarchical SCFQ algorithm with dynamic group weights

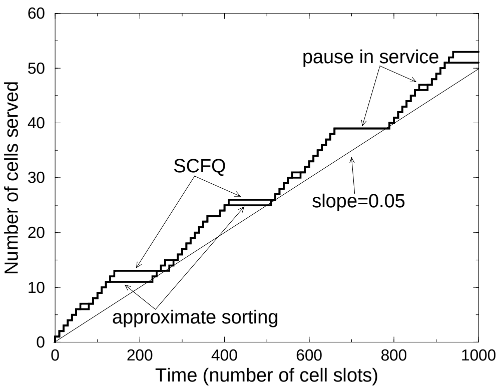

Figure 4: Service for high-rate connection ($r = 1/3$)

small time scale. All connections are continually backlogged starting at time 0. For both the exact and approximate (cheaper) architectures, each high-rate connection receives 5% of the link bandwidth, as seen by the slope of the plots in Figure 4. Performance for the medium-rate and low-rate connections is very similar between the two SCFQ schemes.

The high-rate connections receive high throughput in some time intervals, followed by small idle periods, as shown in Figure 4. These idle intervals occur because multiple connections have the same virtual finishing time. For example, starting at time 150, the server consumes 100 consecutive time slots transmitting one cell from each connection with weight 1/40; similarly, the smaller service gaps correspond to the transmission of one cell from each of the 20 connections with weight 1/24. Although this forces the high-rate connections to wait longer for service, these disruptions occur on a relatively small time scale, bounded by the SCFQ algorithm. The idle gaps occasionally occur slightly earlier under the approximate sorting scheme, since the server rounds F values, sometimes placing a high-rate connection's next cell in a later sorting bin.

The next experiment considers how the architectures affect the burstiness of the high-rate connections. The high-rate connections generate constant bit rate (CBR) traffic, with one cell every 25 slots ($\rho = 0.04$); the ten connections have random starting times, uniformly distributed from 0 to 24. The connections with weight 1/24 create cells according to an on/off source with $\sigma = 10$ and $\rho = 0.005$; each connection has a random starting time, uniformly distributed throughout the length of an on/off period (2200 cell slots). The low-rate connections, with weights 1/40, generate bursty traffic according to a on/off source, with leaky-bucket parameters $\sigma = 50$ and $\rho = 0.003$, resulting in an on/off period of 17034 cell slots. Together, the 130 connections demand 80% of the link bandwidth. The experiment varies the behavior of the low-rate connections, with weight 1/40, to test the SCFQ architectures under varying degrees of congestion. For the low-rate connections, starting times are uniformly distributed within an interval, where the interval length is varied from 0 to 17034, as represented by the x-axis in Figure 5; smaller values of x generate periods of heavier congestion.

Figure 5 shows how congestion affects the burstiness parameters σ^{out} and $\overline{\sigma}^{out}$ of the outgoing cell streams. This distortion increases when more low-rate connections are active simultaneously. The approximate sorting induces slightly larger σ^{out} values than the exact SCFQ scheme, since rounding virtual finishing times introduces additional variability in a connection's service rate. Similarly, the approximate sorting scheme causes slightly larger values for the av-

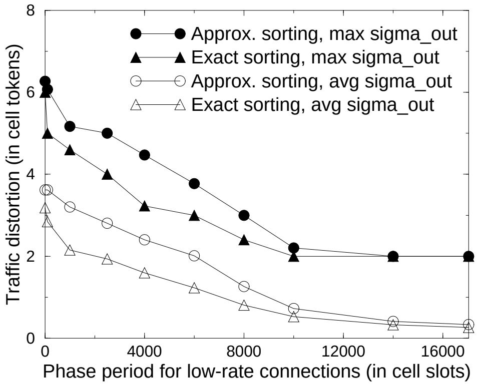

Figure 5: Traffic distortion for high-rate connections

erage traffic distortion $\overline{\sigma}^{out}$. In contrast, experiments with a traditional FIFO server resulted in σ^{out} and $\overline{\sigma}^{out}$ values in the *hundreds*.

5.3 Hierarchical Arbitration

As described in Section 4, the hierarchical SCFQ architectures efficiently support a wide range of connection throughput requirements. In this experiment, one group consists of four backlogged connections with weight 1, while another group handles forty connections with weight 1/10. This experiment taxes the ability of dynamic group weights to adjust to incremental changes in traffic load by having each low-rate connection become idle after transmitting ten cells, except for one connection that remains active throughout the simulation. Figure 6 shows this connection's performance, as well as the service received by one of the high-rate connections. As low-rate connections become idle, starting around time 800, the static-weights scheme assigns all of this excess bandwidth to the remaining low-rate connections. This, in turn, permits more low-rate connections to complete transmitting their tenth cell, resulting in a large boost in service to the lone backlogged connection. In contrast, the dynamic-weights scheme divides this excess bandwidth among both groups, nearly doubling the service rate for connections in the high-rate group.

However, dynamic hierarchical arbitration lags behind exact SCFQ in serving the lone low-rate connection after time 800. To explain this phenomenon, consider the transmission of the last cell from one of the other 39 low-rate connections. If this connection had another cell, its virtual finishing time would be larger than the next cell from each of the other connections in the group; hence, this cell would not affect the relative service order of the next cell from the remaining low-rate connections. However, this extra cell would improve the actual service *time* of these other cells by maintaining the group weight; instead, the connection becomes idle and the group weight decreases, increasing the time until the group's next service event. After this transient change in the group weight, the lone low-rate connection starts receiving service at the appropriate rate, as seen by the slope of the service curves.

5.4 Wide Range of Connection Rates

Despite these transient effects, hierarchical arbitration with dynamic weights delivers good performance and significantly reduces implementation cost, especially when the server must handle a wide range of connection bandwidth requirements. This subsection considers SCFQ servers that handle throughput requirements ranging from 0.004% to 20% of the link bandwidth. In this experiment, each connection generates traffic according to a on/off source, based on the leaky-bucket parameters shown in Table 1. Connections have random starting times, uniformly distributed throughout the length of their on/off periods; this creates a balanced traffic load of 94% link utilization throughout the simulation.

An exact implementation of self-clocked fair queueing requires 15001 FIFO bins to arbitrate among the competing connections, with weights $\{1/3, 1/4, 1/3000, 1/15000\}$ proportional to connection ρ values. In contrast, a hierarchical architecture could divide the four sets of connections into two groups, with weights $\{1/3, 1/4\}$ and $\{1/3000, 1/15000\}$, respectively. The high-rate group requires 5 ($= 1 + 1/\frac{1}{4}$) sorting bins, while the other group requires 6 bins, using weights $\{1/1, 1/5\}$ to differentiate between the two sets of low-rate connections. Table 1 compares this hierarchical arbiter to a calendar queue arbiter with the same number of sorting bins; with such a large range of connection rates, each of the eleven sorting FIFOs must handle a range of 1500 virtual finishing times.

These coarse-grain sorting bins degrade the traffic characteristics of the incoming connections, as shown in Table 1(a); the table indicates the worst-case and average traffic distortion, using the same performance metrics as Figure 5. Distortion is worst for the high-rate connections, since they are most sensitive to service delays and rounding error; this disrupts the incoming CBR traffic by transmitting a burst of cells, possibly increasing cell delay and loss rates at downstream nodes. Still, the approximate sorting scheme performs much better than a traditional FIFO server, which introduces significant traffic distortions.

Interestingly, the dynamic hierarchical arbitration scheme introduces noticeably *less* distortion, even

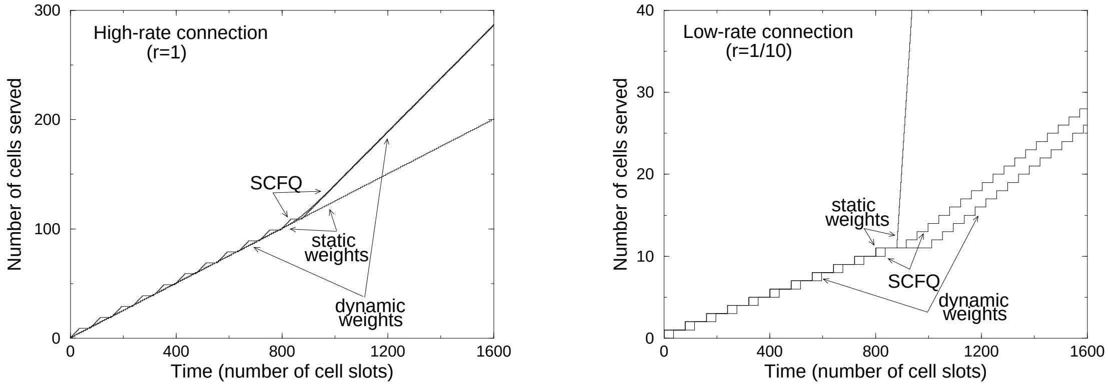

Figure 6: Cell service as multiple low-bandwidth connections become idle

compared to the exact implementation of self-clocked fair queueing. This occurs because the hierarchical arbiter assigns low-rate and high-rate connections to different FIFO sorting bins. Under exact SCFQ, a high-rate connection may wait behind a long FIFO of low-rate traffic before receiving service, accumulating a burst of waiting cells. In contrast, hierarchical arbitration limits the number of low-rate connections that can receive service between visits to the high-rate group, providing a minimum bandwidth to the high-rate connections on a smaller time scale. This effect is quite significant when multiple low-rate connections are active simultaneously, as shown in Table 1(b). In this experiment, the 1000 connections with weight 1/15000 have their start times uniformly distributed in an interval of length 100000. The exact and approximate SCFQ schemes significantly distort the high-bandwidth traffic, while the hierarchical arbitration scheme insulates the high-rate connections from the heavy congestion in the low-rate group.

6 Conclusions

Effective cell scheduling algorithms permit high-speed switches to divide link bandwidth fairly among competing connections. This guaranteed service rate enables fair scheduling algorithms, such as self-clocked fair queueing, to preserve connection traffic descriptors and satisfy quality-of-service requirements. Ultimately, fair queueing algorithms require compromises between implementation complexity and accuracy in approximating an idealized link-sharing model. The SCFQ architectures in Sections 3 and 4 have efficient hardware implementations and apportion bandwidth fairly on a relatively small time scale.

As future work, we are analyzing the properties of the approximate SCFQ architectures, including the hierarchical scheme with dynamic group weights. Further simulation experiments, involving multi-hop networks, should lend greater insight into the worst-case performance of these arbitration schemes, relative to generalized processor sharing and exact SCFQ. Preliminary simulation results suggest that dynamic hierarchical arbitration can actually outperform exact SCFQ, particularly when the server handles a large number of connections with a wide range of bandwidth requirements.

We are also investigating alternate algorithms for adjusting group weights in the hierarchical SCFQ scheme; more conservative approaches to decreasing group weights may provide better guaranteed performance at the connection level. Related algorithms can adjust *connection* weights to adapt to the changing bandwidth requirements of available bit rate (ABR) traffic. With efficient hardware implementations, fair queueing algorithms can play an important role in emerging ATM networks.

References

[1] J. Nagle, "On packet switches with infinite storage," *IEEE Trans. Communications*, vol. 35, pp. 435–438, April 1987.

[2] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *J. Internetworking: Research and Experience*, pp. 3–26, September 1990.

[3] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single node case," *IEEE/ACM Trans. Networking*, vol. 1, June 1993.

[4] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The multiple node case,"

Traffic Pattern				FIFO		GPS		SCFQ		Approx.		Hierarch.	
N	r	ρ	σ	σ^{out}	$\overline{\sigma}^{out}$	σ^{out}	$\overline{\sigma}^{out}$	σ^{out}	$\overline{\sigma}^{out}$	σ^{out}	$\overline{\sigma}^{out}$	σ^{out}	$\overline{\sigma}^{out}$
1	1/3	0.20000	0	149.0	127.3	0.0	0.0	3.0	1.5	5.0	3.2	2.0	0.2
4	1/4	0.15000	0	106.8	90.9	0.0	0.0	2.4	0.9	3.2	1.6	2.0	0.1
500	1/3000	0.00020	100	101.2	0.0	100.0	0.0	101.0	0.0	101.0	0.0	101.0	0.0
1000	1/15000	0.00004	100	101.0	0.0	100.0	0.0	101.0	0.0	101.0	0.0	101.0	0.0

(a) Low congestion (phase interval of 2525000 for low-rate connections)

Traffic Pattern				FIFO		GPS		SCFQ		Approx.		Hierarch.	
N	r	ρ	σ	σ^{out}	$\overline{\sigma}^{out}$	σ^{out}	$\overline{\sigma}^{out}$	σ^{out}	$\overline{\sigma}^{out}$	σ^{out}	$\overline{\sigma}^{out}$	σ^{out}	$\overline{\sigma}^{out}$
1	1/3	0.20000	0	17851.0	252.1	0.0	0.0	55.7	49.1	217.7	198.0	2.0	0.3
4	1/4	0.15000	0	12851.0	302.6	0.0	0.0	35.7	30.6	96.2	88.1	2.0	0.2
500	1/3000	0.00020	100	113.5	0.6	100.0	0.0	102.0	0.0	101.9	0.0	102.0	0.0
1000	1/15000	0.00004	100	101.0	0.0	100.0	0.0	76.6	0.0	76.5	0.0	76.5	0.0

(b) High congestion (phase interval of 100000 for low-rate connections)

Table 1: Traffic distortion under a wide range of connection rates (high congestion)

IEEE/ACM Trans. Networking, vol. 2, pp. 137–150, April 1994.

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

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

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

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

[9] S. J. Golestani, "Network delay analysis of a class of fair queueing algorithms," *IEEE Journal on Selected Areas in Communications*, vol. 13, pp. 1057–1070, August 1995.

[10] J. W. Roberts, "Virtual spacing for flexible traffic control," *International Journal of Communication Systems*, vol. 7, pp. 307–318, October–December 1994.

[11] L. Zhang, "Virtual Clock: A new traffic control algorithm for packet-switched networks," *ACM Trans. Computer Systems*, vol. 9, pp. 101–124, May 1991.

[12] G. G. Xie and S. S. Lam, "Delay guarantee of virtual clock server," *IEEE/ACM Trans. Networking*, vol. 3, pp. 683–689, December 1995.

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

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

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

[16] D. Saha, S. Mukherjee, and S. K. Tripathi, "Multi-rate traffic shaping and end-to-end performance guarantees in ATM," in *Proc. IEEE International Conference on Network Protocols*, October 1994.

[17] 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, November 1992.

[18] 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, September 1992.

[19] E. Wallmeier and T. Worster, "The Spacing Policer, an algorithm for efficient peak bit rate control in ATM networks," in *Proc. International Switching Symposium*, pp. 22–26, October 1992.

[20] S. Shenker, D. D. Clark, and L. Zhang, "A scheduling service model and a scheduling architecture for an integrated services packet network." Working paper, Xerox PARC, August 1993.

[21] S. Floyd and V. Jacobson, "Link-sharing and resource management models for packet networks," *IEEE/ACM Trans. Networking*, vol. 3, pp. 365–386, August 1995.

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

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