Scalable Architectures for Integrated Traffic Shaping and Link Scheduling in High-Speed ATM Switches

Jennifer Rexford, Flavio Bonomi, Albert Greenberg, and Albert Wong

Abstract— **Emerging broadband switches must accommodate the diverse traffic parameters and quality-of-service requirements of voice, data, and video applications. End-to-end performance guarantees depend on connections complying with traffic contracts as their cells travel through the network. This paper presents a leaky-bucket shaper architecture that scales to a large number of connections with diverse burstiness and bandwidth parameters. In contrast to existing designs, the proposed architecture arbitrates *fairly* between connections with conforming cells by carefully *integrating* leaky-bucket traffic shaping with rate-based scheduling algorithms. Through a careful combination of per-connection queueing and approximate sorting, the shaper performs a small, bounded number of operations in response to each arrival and departure, independent of the number of connections and cells. When the shaper must handle a wide range of rate parameters, a hierarchical arbitration scheme can reduce the implementation overheads and further limit interference between competing connections. Through simulation experiments, we demonstrate that the architecture limits cell shaping delay and traffic distortions, even in periods of heavy congestion. The efficient combination of traffic shaping and link scheduling results in an effective architecture for managing buffer and bandwidth resources in large, high-speed ATM switches.**

Keywords— **Asynchronous transfer mode, broadband communication, switches, scheduling, traffic control**

I. Introduction

The advent of integrated networks for voice, data, and video applications introduces new challenges in supporting performance guarantees. With high-speed links and small cell sizes, modern ATM (asynchronous transfer mode) switches require efficient hardware to process cell arrivals and departures every few microseconds, if not faster; in addition, these architectures should scale to a large number of connections with diverse traffic parameters and quality-of-service requirements. End-to-end guarantees for delay, throughput, and loss depend on the successful provisioning of buffer and bandwidth resources in the network, based on traffic contracts established during admission control. To regulate connections and avoid buffer overflow, broadband networks can employ traffic *shaping* to delay incoming cells until they conform to connection burst and bandwidth descriptors.

Emerging broadband networks introduce new challenges in designing high-speed shaper architectures that can scale to a large number of connections with diverse traffic parameters. Many new networking applications, such as large-scale web or video servers, require traffic shaping for hundreds or even thousands of connections with different burst and bandwidth descriptors. ATM switches can also amortize implementation costs across multiple end systems by providing traffic shaping as a service at the network edge. In addition to shaping at the network end points, available-bit-rate connections require traffic enforcement in the *interior* of the network at each virtual source node, where a connection's bandwidth allocation may change over time in response to feedback from the network [1]. Variable-bit-rate and constant-bit-rate connections may also be reshaped in the interior of the network to limit delay variation and buffer requirements at the downstream switches [2–5]. Reinforcing the traffic parameters is particularly important when connections traverse switches with different performance characteristics or service providers.

Most existing traffic shapers [6–11] employ some version of *leaky-bucket* control to buffer non-conforming cells and schedule them for later transmission. Conceptually, a leaky-bucket controller generates credit tokens at rate ρ, where the token bucket holds at most σ credits [12]; an arriving cell must claim a token before receiving service. Given the status of the token bucket for each connection, the shaper can determine the conformance time of each arriving cell [13] and arbitrate access to the outgoing link. Conflicts can arise when multiple cells, from different connections, become eligible for transmission during the same time slot. As a result, the shaper can develop a *backlog* of conforming cells, particularly when traffic arrives from multiple input links or a single high-speed link. Depending on how the switch arbitrates amongst conforming cells, collisions can distort connection leaky-bucket parameters and increase cell shaping delays, even for cells that are conforming on arrival. As a result, the outgoing cells may violate the traffic descriptors expected by downstream switches, possibly increasing delay and loss; this is especially problematic for constant-bit-rate and real-time variable-bit-rate connections, which have a low tolerance for delay variation.

Minimizing these distortions requires a link-scheduling policy that controls the interference between competing connections. In contrast to first-in first-out scheduling, fair arbitration schemes [14–21] can limit traffic distortions by guaranteeing that each backlogged connection receives its share of the link bandwidth on a small time scale. If the incoming traffic were *already* leaky-bucket compliant, weighted fair scheduling would ensure that multiplexing the connection with other traffic would never inflate the burst parameter σ by more than one cell [15], guaranteeing well-behaved input traffic for the next switch in the route. However, traffic shaping requires the switch to handle a *mixture* of conforming and non-conforming cells. The switch can develop backlogs of both conforming and non-conforming traffic, as shown in Fig. 1; in fact, a spike in

J. Rexford and A. Greenberg are with AT&T Labs Research in Florham Park, New Jersey; F. Bonomi is with ZeitNet/Cabletron in Santa Clara, California; and A. Wong is with Lucent Technologies in Holmdel, New Jersey.

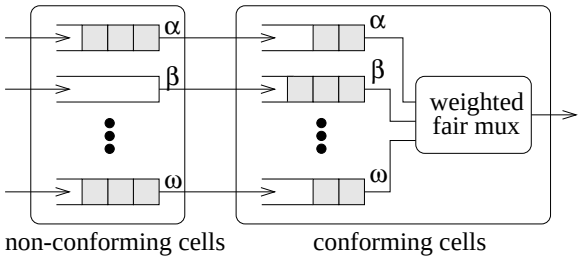


Fig. 1. **Fair Traffic Shaping:** Conceptually, a fair traffic shaper divides link bandwidth between connections with conforming cells, based on their rate parameters. However, this idealized model requires separate data structures for non-conforming and conforming cells; in addition, multiple cells may move between the two data structures in one time slot, further complicating the implementation.

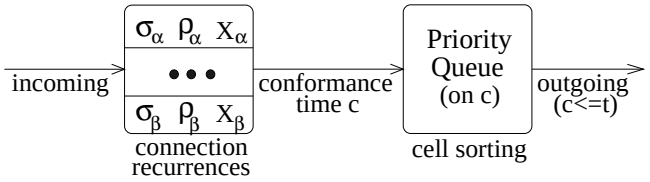


Fig. 2. **Leaky-Bucket Traffic Shaper:** To efficiently handle a large number of connections, a traffic shaper can consist of a table for storing the traffic parameters and the leaky-bucket status of each connection, followed by a priority queue that ranks waiting cells by their conformance times.

the non-conforming backlog can rapidly become a spike in the conforming backlog, if multiple cells reach conformance in a small interval of time. These properties introduce new challenges in designing effective mechanisms that combine fair link scheduling with traffic shaping, while scaling to thousands of connections with diverse leaky-bucket parameters.

This paper presents a practical shaper architecture that *integrates* leaky-bucket traffic shaping and rate-based link scheduling to reduce both implementation complexity and traffic distortions [22]. As a result, the architecture can serve as a leaky-bucket shaper at the ingress or egress of the network, with a fair division of link bandwidth during transient overloads of conforming traffic. Alternatively, the architecture can be viewed as a non-work-conserving, rate-controlled link scheduler [2–5] for use in the interior of the network to reduce cell delay jitter and downstream buffer requirements. After a review of existing shaper designs in Section II, Section III describes how the proposed architecture combines per-connection queueing, approximate sorting algorithms, and hierarchical arbitration. Section IV extends this architecture to minimize traffic distortions. Instead of simply *concatenating* the traffic shaping and fair multiplexing operations, as in Fig. 1, the architecture *integrates* rate-based scheduling into the shaping logic. The simulation experiments in Section V demonstrate that the shaper scales to a large number of connections with diverse traffic descriptors, even under heavy congestion. Section VI concludes the paper with a discussion of future research directions.

This paper complements existing work on traffic shaper architectures for high-speed networks [6–11] by emphasizing implementation and performance scalability. Ongoing research on rate-based link-scheduling algorithms [14–21] motivates the application of weighted fair arbitration to improve the performance properties of the traffic shaper, as discussed in Section IV. Recent research on rate-based scheduling algorithms considers schemes that improve fairness by temporarily "stalling" busy connections that have received link bandwidth ahead of schedule [23–25]. In contrast to our leaky-bucket shaper, these link-scheduling schemes define cell eligibility in terms of the underlying fair queueing discipline, without necessarily enforcing burst and bandwidth parameters for each connection. The hierarchical arbitration scheme, analyzed and discussed in Sections III-C and IV-B, extends our earlier work on efficient link-scheduling algorithms [26]. The contribution of this paper is a set of effective architectural techniques for *integrating* hierarchical arbitration and fair link scheduling with leaky-bucket traffic shaping.

II. Review of Shaper Architectures

High-speed shaper designs require simple and efficient techniques for recognizing when cells conform to their connections' traffic descriptors. Scalable architectures typically employ per-connection recurrences to assign a conformance time to each arriving cell, followed by a sorting unit that schedules cells for departure, as shown in Fig. 2.

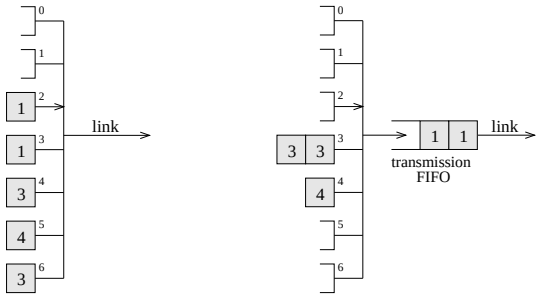
A. Connection Recurrences

ATM switches could monitor and regulate connections by *policing* the incoming cell streams. When an arriving cell violates its connection's contract, a policer either discards the "non-conforming" cell or marks the cell as low-priority traffic, permitting downstream switches to drop the excess cell when the network is congested. 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. However, 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. This would require end-to-end protocols to either avoid short-term violations of connection traffic parameters or include efficient mechanisms to recover from cell loss.

Instead of dropping or marking non-conforming cells, a traffic *shaper* delays incoming cells until they conform to the connection's burst and bandwidth descriptors, at the expense of increased implementation complexity. Although a traffic shaper could conceivably maintain a separate leaky-bucket controller and cell buffer for each admitted connection, a scalable design requires a more integrated approach [6]. Instead, the shaper can maintain a table that stores the leaky-bucket status of each connection. The shaper uses this information to compute the time c that an arriving cell first conforms to its connection's traffic descriptors. When a new cell arrives, the shaper identifies the

```
X = X + 1/ρ;              // Estimated cell arrival time
if (X ≤ t)                // Full token bucket: reset X
   c = X = t;
else if (X ≤ t + σ/ρ)     // Partially full token bucket
   c = t;
else                      // Empty token bucket: delay cell
   c = X − σ/ρ;
```

Fig. 3. **Computing Cell Conformance Times:** An efficient algorithm can compute the conformance time c for an incoming cell, based on its estimated arrival time X and its actual arrival time t. Using the connection leaky-bucket parameters σ and ρ, the algorithm computes X to determine the current status of the token bucket; initially, $X = -1/\rho$ to ensure that $X = 0$ after the first cell arrival.



(a) One cell per bin (b) Linked list per bin

Fig. 4. **Ranking Cells in Sorting Bins:** A traffic shaper can rank waiting cells using a collection of sorting bins, where each bin corresponds to a time slot. In (a), an arriving cell locates the first empty bin at or after its conformance time c. In contrast, the shaper in (b) places all cells with the same c value in a single bin; a separate linked list holds any conforming cells awaiting service. In the figure, the current time is $t = 2$ and each cell (shaded) indicates its conformance time.

appropriate connection and updates its state. The shaper can limit its tolerance for non-conforming traffic by discarding an incoming cell if it has a large conformance time (i.e., $c - t$ larger than some threshold), particularly when buffer space is limited.

Shapers can employ a variety of algorithms to compute cell conformance times, based on one or more traffic descriptors. For example, Fig. 3 shows a leaky-bucket shaping scheme based on the virtual scheduling technique in ATM Forum's generic cell rate algorithm [13]. The shaper assigns a conformance time to each incoming cell, using a state variable X that represents the cell's *estimated* arrival time, based on the connection's shaping rate ρ. A cell arriving at most σ/ρ time units ahead of X can still claim a token; otherwise, the cell must wait until the controller generates an additional token for the connection at time $X - \sigma/\rho$. To avoid clock wrap-around errors, the shaper can periodically update each connection's state, independent of cell arrivals; in particular, if an idle connection has accumulated a full token bucket (i.e., $X \leq t$), the shaper can flag the connection's table entry to reset X on the next cell arrival. For a connection with no burst tolerance (i.e., $\sigma = 0$), the algorithm reduces to

$$c = X = \max\{X + 1/\rho_{peak}, t\}$$

to ensure that consecutive cells have c values spaced by at least $1/\rho_{peak}$ time units. The shaper can enforce both peak and sustainable cell rates by using a dual leaky-bucket algorithm to assign conformance times, resulting in a $(\sigma, \rho, \rho_{peak})$ specification for each connection, with peak rate $\rho_{peak} \geq \rho$.

B. Cell Sorting

The shaper uses the cell conformance times c to schedule traffic for transmission. One approach employs a priority queue to rank cells in order of increasing c values, as shown in Fig. 2; during each transmission slot, the shaper transmits the queue's head cell if its conformance time has been reached (i.e., $t \geq c$). High-speed operation typically requires hardware support for cell scheduling. For example, the priority queue could consist of a large shift register that keeps the cells (or pointers to the cells) in sorted order [7]. To enqueue an incoming cell with conformance time c, each element in the shift register compares its key to c; any cells with larger conformance times shift by one location to make room for the new cell. However, the hardware costs of the shift register increase with the number of elements, due to the parallel comparison operations; in addition, handling clock rollover requires extensions to the design, to correctly differentiate between large values of c (before rollover) and small values of c (after rollover) [6, 27].

To avoid the complexity of an exact priority queue implementation, a shaper could dynamically construct a link transmission schedule as each cell arrives. In this approach, the schedule consists of a large collection of bins, where each bin corresponds to a single transmission slot [8], as shown in Fig. 4(a). The shaper sequences through the bins at the link rate, transmitting a cell in each time slot unless the current bin is empty. An arriving cell enters the first empty location at or after its conformance time c, modulo the number of bins; this requires the design to include complex logic for locating empty bins. The shaper drops the incoming cell if $c - t$ exceeds the range of the bins. When the scheduler services a large number of connections, under heavy load, the shaper may have to schedule a cell for transmission much later than its conformance time c. The insertion mechanism favors earlier arrivals, possibly delaying a well-behaved connection to serve other cells with larger conformance times.

To transmit cells in order of increasing conformance times, the shaper can employ a *calendar queue* [28], as shown in Fig. 4(b). Instead of storing a single cell in each bin, this architecture maintains a logical *linked list* of cells at each conformance time [9]. Although this requires additional memory to store a pointer field with each cell, an arriving cell can join the linked list that corresponds to its c value without searching for an empty slot in the schedule. Since the link cannot transmit more than one cell in each time slot, the shaper maintains a separate FIFO for holding any cells that have reached their conformance times; in each time slot, the shaper appends this transmission FIFO

with the contents of the current sorting bin. Hence, under a backlog of conforming traffic, the shaper transmits cells in order of increasing c values.

Although the sorting bins obviate the need for a complex priority queue, cells may have c values ranging far into the future, particularly when a low-bandwidth connection has a backlog of non-conforming traffic. This introduces a fundamental trade-off between the number of sorting bins and the likelihood of dropping a non-conforming cell. To reduce the required number of sorting bins, the shaper could associate each bin with a range of g consecutive conformance times; for example, if $g = 5$, a single bin would handle cells that have $c \in \{0, 1, 2, 3, 4\}$. However, shaping with a large bin granularity can introduce excessive jitter that distorts the leaky-bucket parameters of the outgoing traffic, particularly for high-bandwidth connections. In addition, the FIFO service in existing shaper architectures does not control the interference between competing connections when multiple conforming cells await service. The next two sections propose a scalable shaper architecture that addresses these implementation and performance challenges.

III. Scalable Leaky-Bucket Traffic Shaper

Emerging high-speed networks require traffic-shaping and link-scheduling architectures that can handle a large number of cells from connections with a wide range of bandwidth parameters. This section introduces a new shaper architecture that limits implementation complexity by combining per-connection queueing, approximate sorting, and hierarchical arbitration.

A. Per-Connection Queueing and Approximate Sorting

Although a connection α may have a large backlog of cells, with a wide range of conformance times, the connection's *head-of-line* cell has a c value at most $1/\rho_\alpha$ time slots into the future; otherwise, the connection's previous cell would still reside in the shaper. Hence, the shaper can reduce the required number of sorting bins by considering only the head-of-line cell from each backlogged connection, while holding the connection's remaining cells in a FIFO linked list. In addition to limiting sorter complexity, per-connection queueing enhances architectural flexibility by facilitating effective buffer-management and link-scheduling policies, as well as rapid responses to changes in connection rate parameters. Although per-connection queueing can complicate the implementation of exact sorting, approximate schemes can significantly reduce hardware complexity and provide fair service to the busy connections.

To illustrate the difficulty of implementing exact sorting with per-connection queueing, consider the situation when a cell enters the sorting unit upon the departure of the connection's previous cell. If the shaper has accumulated a backlog of conforming traffic, this new head-of-line cell may have *already* reached its conformance time (i.e., the cell may have $c < t$). Then, to transmit cells in order of their c values, the shaper would have to insert this new head-of-line cell *in between* other conforming cells awaiting service. To handle these cells, the sorting unit can include extra data structures to hold conforming traffic, in addition to the bins for the non-conforming head-of-line cells [11]. However, this introduces a potentially large number of additional sorting bins, as well as more complicated logic for sequencing through the data structures. Also, ranking conforming cells by their c values complicates the effort to avoid per-cell timestamps, as discussed in Appendix A.

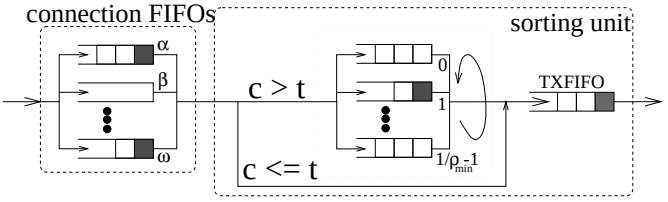


Fig. 5. **Scalable Shaper Architecture:** The proposed shaper architecture consists of per-connection FIFO queues, followed by a sorting unit that consists of a small collection of sorting bins (with grain $g = 1$ in this example) and a transmission FIFO for conforming cells. A cell enters the sorting unit upon arrival to an idle connection or on the departure of the connection's previous cell; a conforming head-of-line cell proceeds directly to the transmission FIFO, whereas a non-conforming head-of-line cell joins the sorting bin based on its conformance time c.

Approximate sorting algorithms may be necessary to avoid these overheads in high-speed shaper implementations. Instead of exact sorting, a shaper can reduce implementation complexity by placing all conforming head-of-line cells in a single transmission FIFO, as shown in Fig. 5. Since the sorting bins hold only non-conforming head-of-line cells, the shaper requires at most $b = 1/(g\rho_{\min})$ bins, independent of the number of connections or the amount of backlogged traffic, where g is the bin granularity. Every g time slots, the shaper concatenates a single sorting bin with the transmission FIFO, as in Fig. 4(b). After transmitting a cell, the connection's next cell proceeds to a sorting bin (if $c > t$) or directly to the transmission FIFO (if $c \leq t$), as shown in Fig. 6. If the new head-of-line cell has already reached its conformance time, this policy sacrifices accuracy in order to reduce implementation complexity. When connections have similar ρ values, this approximate scheme can actually *outperform* exact sorting by guaranteeing a minimum service rate to each connection on a small time scale, as shown in Section V-B.

B. Implementation Complexity

Fig. 7 highlights the main data structures in the approximate shaper architecture, with s cells, v connections, and b sorting bins. This figure shows a single outgoing link that services a serialized stream of cells from one or more incoming links; in a larger switch, multiple outgoing links may share access to a common cell memory [29]. The shaper consists of $v + b + 1$ logical linked lists that represent the connection FIFOs, the sorting bins, and the transmission FIFO, where each list consists of a head pointer, a tail pointer, and an empty flag; each connection FIFO includes an additional flag to indicate whether or not the connection has a head-of-line cell in the sorting unit. To connect

Cell Arrival

```
compute c for the new cell;
if (connection is currently idle) {
      // New head-of-line cell
      if (c == t) // Conforming cell
            enqueue cell onto transmission FIFO;
      else // Non-conforming cell
            enqueue cell onto bin ⌊(c mod bg)/g⌋;
}
else // Connection already backlogged
      enqueue cell onto connection FIFO;
```

Cell Departure

```
if (connection is still backlogged) {
      dequeue cell from connection FIFO;
      // New head-of-line cell
      if (c ≤ t) // Conforming cell
            enqueue cell onto transmission FIFO;
      else // Non-conforming cell
            enqueue cell onto bin ⌊(c mod bg)/g⌋;
}
else // Connection now idle
      // No action for idle connection
```

Conforming Bin

```
if (t mod g == g − 1)
      append bin ⌊(t mod bg)/g⌋ onto transmission FIFO;
```

Fig. 6. **Shaping Algorithm with Per-Connection Queueing:** The proposed shaper architecture performs a small, bounded number of operations in response to each arriving and departing cell. The sorting unit consists of b bins of grain g, as well as a transmission FIFO.

cells in the various linked lists, each buffered cell includes a small pointer field that consists of $\lceil \log_2 s \rceil$ bits to denote a location in the cell memory. The shaper also includes a free list for assigning unused memory locations to arriving cells; initially, this list includes every element in the memory, with each entry indicating the next location in the buffer. To simplify operation, the shaper maintains the idle addresses as a last-in first-out list, using the pointer fields associated with each location in the cell buffer, as shown in Fig. 7. Upon cell departure, the shaper returns the free memory location to the head of the free list.

As shown in Fig. 6, each cell arrival or departure introduces a small, bounded number of enqueue and dequeue operations on these linked lists, independent of s and v. The shaper can use simple pointer manipulations to add and delete entries in the logical FIFO queues [30]. If the design includes separate memory modules for each data structure, as shown in Fig. 7, the shaper can overlap operations on different units to achieve greater concurrency in processing cell arrivals and departures. Depending on the link speed and the size of the various data structures, the cell buffer may employ a slower memory technology than the other units, for reduced cost, area, and power requirements. Through per-connection queueing and approximate sorting, the architecture tightly bounds the number of the scheduling bins, independent of the number of cells and the number of connections. As discussed in Appendix A, the approximate sorting scheme also allows the switch to avoid the overhead of storing the conformance time c of each cell; this additional optimization is not possible under exact sorting.

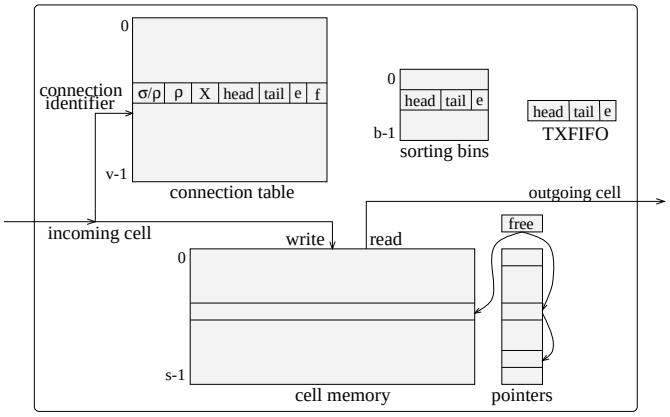


Fig. 7. **Efficient Hardware Implementation:** The proposed shaper architecture has an efficient hardware implementation that supports v connections and s cells, with b sorting bins. The buffered cells each include a pointer field for constructing logical linked lists in the connection FIFOs, the sorting bins, and the transmission FIFO. A separate memory stores the cells awaiting service.

C. Hierarchical Architecture With Bandwidth Groups

Although per-connection queueing can reduce scheduling complexity in the shaper, handling a wide range of connection rate parameters still requires a large number of sorting bins. For a low-rate connection, even the head-of-line cell could have a conformance time far into the future. Although low-bandwidth traffic requires the sorting unit to handle a wide *range* of c values, these connections can tolerate some inaccuracy in cell scheduling, permitting coarse-grain sorting bins. On the other hand, a large bin granularity g can introduce significant shaping delay and jitter to high-rate connections. To reconcile these conflicting requirements, the shaper can group connections based on their bandwidth requirements, allowing each sorting unit to select a different grain (g) and range (bg) for its bins. As shown in Fig. 8, this results in a two-level architecture, where each group consists of a sorting unit that handles a large number of connections with similar rates; an additional arbiter services the small number of groups with diverse bandwidth requirements as discussed in Section IV.

With a hierarchical architecture, the shaper can select a small sorting granularity (g) for high-rate connections to reduce delay and jitter, relative to the existing architectures in Section II. To formalize these trade-offs, consider a shaper with connection bandwidth parameters that can vary from $\rho_{\min}$ to $\rho_{\max}$, where each group handles rates within a factor $m > 1$ of each other. As a result, the shaper

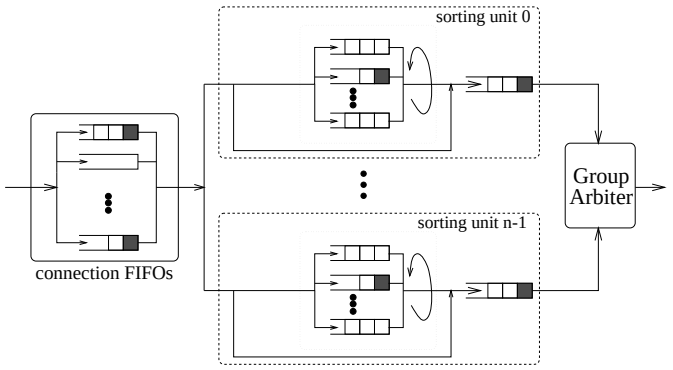


Fig. 8. **Hierarchical Shaper Architecture:** In the hierarchical architecture, the shaper consists of connection FIFOs and n sorting units that each consist of sorting bins and a transmission FIFO. Each sorting unit i tailors its bin granularity g_i to a small range of connection bandwidth parameters.

consists of $n = \log_m\{\rho_{\max}/\rho_{\min}\}$ groups, where any connections $\rho_{\alpha_i} \in [m^i\rho_{\min}, m^{i+1}\rho_{\min})$ belong to group i, for $i = 0, 1, \ldots, n-1$; for example, if $m = 16$, the shaper can support rates ranging from 1 kilobit/second to 1 gigabit/second with just *five* groups (since $\log_{16}\{2^{30}/2^{10}\} = 5$). To perform the operations in Fig. 6, the shaper can have dedicated logic for each group or share a single controller across the multiple groups, as discussed in Appendix B. In each group, the *low*-bandwidth connections dictate the range of the sorting bins; for group i, a head-of-line cell can have a c value at most $1/(m^i\rho_{\min})$ time units into the future. To prevent delay and traffic distortions for the high-bandwidth connections, each group should select a bin granularity g_i based on the requirements of its *largest* possible rate; i.e., $g_i = 1/u(m^{i+1}\rho_{\min})$, where a larger value of u corresponds to more precise sorting.

As a result, each group i requires at least

$$b = \frac{\text{range of the bins}}{\text{grain of each bin}} = \frac{1/(m^i\rho_{\min})}{1/u(m^{i+1}\rho_{\min})} = mu$$

sorting bins, for a total of mnu bins in the shaper. As expected, the number of bins increases under more precise scheduling (larger u) or a wider range of connection rates (larger n or m). However, if the shaper did *not* divide connections into groups, a single sorting unit would have to handle the maximum range of $1/\rho_{\min}$ and the minimum granularity of $1/(u\rho_{\max})$, for a total of $m^n u$ bins, where $m^n = \rho_{\max}/\rho_{\min}$. In the example with rates ranging from 1 kilobit/second to 1 gigabit/second, this single sorting group would require over 13,000 times more sorting bins than a hierarchical architecture with $m = 16$, for the same value of u. Hence, a hierarchical architecture has the potential to significantly reduce the memory requirements of the shaper's sorting logic, particularly when connections have diverse rate parameters. Alternatively, for the same number of sorting bins, the hierarchical architecture can support much more precise scheduling for the high-rate traffic; this can significantly reduce cell shaping delay and jitter, relative to existing approaches.

IV. Integrating Fair Link Scheduling

The architecture in Fig. 8 scales well with the number of connections, the number of cells, and the range of connection rate parameters. However, good performance depends on how well the switch arbitrates between connections when multiple conforming cells await service. By carefully multiplexing cells from different connections, the switch can ensure that each connection receives a fair share of the link bandwidth on a small time scale, even during periods of heavy congestion.

A. Fair Arbitration Within a Group

Ideally, a traffic shaper schedules an arriving cell for transmission at its conformance time c, forwarding conforming cells directly to the outgoing link. However, since the link carries traffic for multiple connections, several cells may become eligible for transmission at the same time; as a result, the switch can develop a backlog of conforming traffic, especially during periods of heavy congestion; this is particularly likely in switches with a large number of input ports and connections. As discussed in Section II, most existing traffic shapers transmit conforming cells in order of increasing conformance times. Under a backlog of conforming cells, this "exact sorting" favors connections with larger σ values since these connections can have multiple cells in a small range of c values. These bursts restrict link access for connections with smaller σ values; by the time this transient congestion begins to dissipate, a low-σ connection can have a large backlog of conforming traffic, which may generate an unexpected burst on the output link, as shown in Section V-B.

The shaper can mitigate these collision effects by *interleaving* the conforming cells from competing connections, based on the connection rate parameters. The approximate architecture in Fig. 5 suggests an efficient mechanism for fair link scheduling when the shaper consists of a single group. As discussed in Section III, the shaper includes a transmission FIFO for scheduling all conforming head-of-line cells. For connections with a backlog of conforming traffic, each cell transmission triggers the insertion of the connection's next cell into the transmission FIFO. By transmitting cells from this FIFO, the link implicitly sequences through the backlogged connections, effectively providing *round-robin* service to the connections with conforming cells. Consequently, the shaper guarantees a minimum bandwidth to each connection on a relatively small time scale. This reduces worst-case shaping delay and traffic distortions, particularly for connections with small σ values.

However, round-robin arbitration does not provide truly "fair" service when connections have different ρ values, since low-rate and high-rate connections receive the same share of the link bandwidth. Instead, the shaper should employ *weighted* round-robin scheduling, where each connection α receives service based on its bandwidth parameter ρ_α. With a small extension, the architecture in Fig. 5 can achieve weighted round-robin service of conforming cells by allowing connections to insert *multiple* cells into the

sorting unit; in particular, the shaper should allow connection α to have up to $\lfloor \rho_\alpha / \rho_{\min} \rfloor$ consecutive cells in the sorting unit. This allows a high-bandwidth connection to have several conforming cells in the transmission FIFO, ahead of conforming traffic from low-bandwidth connections, even if the low-bandwidth connections have cells with *smaller* c values. When the shaper has a backlog of conforming traffic, this ensures that connections are served in proportion to their bandwidth requirements.

Even though some connections have multiple cells in the sorting unit, the weighted round-robin architecture does not require additional sorting bins, since these cells have conformance times at most $1/\rho_\alpha \cdot \lfloor \rho_\alpha / \rho_{\min} \rfloor$ time units into the future, where the sorting bins handle a range of $1/\rho_{\min}$ values of c. For an efficient implementation of the weighted round-robin service, each connection should maintain a count of the number of cells it has in the sorting unit; in effect, this converts the bit flag f in Fig. 7 into a *counting semaphore* that is incremented (decremented) as the connection's cells enter (leave) the sorting unit. When the count reaches $\lfloor \rho_\alpha / \rho_{\min} \rfloor$, the connection cannot insert another cell into the sorting unit until a cell departs. This results in an efficient architecture that *integrates* traffic shaping and fair link scheduling to ensure that each connection receives sufficient bandwidth on a small time scale.

B. Fair Arbitration Between Groups

The weighted round-robin arbitration can provide fair service to connections in a single group, but the hierarchical shaper architecture requires an effective mechanism for interleaving conforming cells from *different* groups. For an efficient implementation, the shaper can apply fair, rate-based scheduling algorithms to divide link bandwidth between groups; within each group, the weighted round-robin arbitration ensures that each connection receives a "fair share" of the group's bandwidth. Weights ϕ_i coordinate bandwidth sharing between competing groups, where $i = 0, 1, \ldots, n-1$. These weights represent the aggregate bandwidth requirements of each group; in the simplest case, ϕ_i is the sum of ρ_{α_i} across all connections α_i in group i. A variety of rate-based link-scheduling algorithms [14–21] can guarantee that group i receives a fair portion

$$\frac{\phi_i}{\sum_{j=0}^{n-1} \phi_j}$$

of the link bandwidth. The various link-scheduling algorithms differ in terms of implementation complexity and the ability to achieve fairness on a small time scale.

With static weights $\phi_i = \sum \rho_{\alpha_i}$, an idle connection divides its excess bandwidth amongst the busy connections in the same group, instead of sharing with other connections in different groups. This type of hierarchical link-sharing model is particularly useful when groups correspond to different institutions, traffic classes, or protocol families [23, 31, 32], and can ensure that the link guarantees a minimum bandwidth to each connection. As a heuristic extension [26], the hierarchical arbitration could share excess bandwidth between individual connections by altering the group ϕ_i values to reflect the aggregate throughput requirements of the *backlogged* connections in each group over time. That is, a group could assign

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

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) ρ_{α_i} when connection α_i becomes backlogged (empty). By extending existing fair queueing schemes to employ these dynamic weights, the group arbitration scheme can efficiently apportion excess bandwidth more fairly at the connection level, as described in Appendix C.

V. Performance Evaluation

This section focuses on how the various shaper architectures affect cell delay and the leaky-bucket descriptors of the outgoing cell streams. Inherently, the architectures have similar performance under low traffic loads, since the shaper does not develop a significant backlog of conforming traffic. Hence, most of the experiments consider the effects of moderate or heavy congestion on connection quality-of-service parameters, particularly when the shaper handles traffic with different bandwidth and burstiness requirements.

A. Traffic Model and Performance Metrics

The simulation experiments evaluate connections sharing access to a single link that can transmit one cell in each time slot; for stability, the link bandwidth must exceed the sum of the connections' sustainable cell rates. Each connection generates periodic bursts of cells according to an on/off model with leaky-bucket parameters (σ_{in}, ρ_{in}), with a peak-rate equal to the link bandwidth; the burst length σ_{in} could correspond to the packet or message size in a data transmission. The simulation experiments can generate a temporary backlog of non-conforming cells by allowing σ_{in} to exceed the burstiness parameter σ enforced by the shaper. To control the interaction between cell streams, each connection has an independent starting time, uniformly distributed in an interval $[0, x]$; the length of this interval can vary from 0 to the connection's on/off period. Smaller values of x generate periods of heavier congestion and can capture the effects of multiple input links (or one high-speed input link) entering the switch. Although extremely small values of x are not necessarily realistic, varying x enables the experiments to identify performance trends under challenging traffic patterns where a large number of connections turn on or off at nearly the same time.

With this parameterized model of incoming traffic, the simulation experiments can evaluate the performance scalability of the shaper architectures under different arrival patterns (σ_{in}, ρ_{in}), shaping parameters (σ, ρ), and degrees of congestion x. When the shaper accumulates a backlog

of conforming traffic, link arbitration policies can affect the leaky-bucket parameters of the outgoing cell streams. Multiplexing a connection with other network traffic results in an outgoing stream that is (σ_{out}, ρ)-compliant, where σ_{out} may exceed the shaping parameter σ. These traffic distortions violate the expectations of downstream nodes in the network, possibly increasing cell loss rates and end-to-end delay. To evaluate the burstiness of the output stream, the simulator feeds a connection's outgoing cells to a hypothetical link of rate ρ; σ_{out} ($\overline{\sigma}_{out}$) represents the worst-case (average) queue length encountered by the connection's cells. Ideally, the switch produces a well-shaped, (σ, ρ)-compliant stream, resulting in $\sigma_{out} \leq \sigma$. The experiments compare the proposed shaper architecture against a traditional shaper that transmits conforming traffic in order of increasing conformance times (FIFO service of conforming cells). In experiments with multiple bandwidth groups, the shaper employs the dynamic hierarchical arbitration scheme in Fig. 14.

B. Mixing Different Burstiness Parameters

The architecture in Fig. 5 reduces implementation complexity through per-connection queueing and approximate sorting of backlogged conforming cells. Fig. 9 compares the architecture to a shaper that always transmits cells in order of their conformance times. In this experiment, the input traffic consists of 80 connections with large σ values and 10 connections with small σ values, for a total link utilization of 81%:

	Arrival		Shaping	
N	σ_{in}	ρ_{in}	σ	ρ
80	300	0.009	100	0.010
10	50	0.009	5	0.010

To generate periods of congestion in the shaper, the experiment varies the phase of the 80 high-σ connections from 0 to their on/off period, while the low-σ connections have uniform random start times during their on/off period. With an on period of length $\sigma_{in}/(1-\rho_{in})$ and an off period of length σ_{in}/ρ_{in}, the high-σ connections have an on/off period of 33636 time slots, as shown by the x-axis in Fig. 9.

Under exact sorting, this congestion can have an adverse effect on the leaky-bucket parameters of the low-σ connections; the high-σ connections do not experience significant traffic distortion under either shaper architecture. Fig. 9(a) considers the performance of one of the low-σ connections, where the high-σ connections select their start times randomly within one-third of their on/off period (i.e., uniformly distributed in the interval $[0, 11211]$). In Fig. 9(a), the "input traffic" curve shows the on/off pattern of cell arrivals, while the "conforming traffic" curve indicates when these cells become eligible for transmission. Ideally, the link would transmit each cell at its conformance time, but collisions with other connections disrupt this service. Hence, the "output" curves deviate from the "conforming" curve, introducing shaping delay and traffic distortions; the difference along the y-axis represents the connection's backlog of conforming cells, while the difference along the x-axis denotes cell shaping delay.

Exact sorting favors connections with larger σ values, which can inject a burst of cells with a small range of conformance times. In times of congestion, these bursts can restrict link access for connections with smaller σ values; by the time this transient congestion begins to dissipate, a low-σ connection can have a large backlog of conforming traffic, which can then generate an unexpected burst on the output link. This can introduce significant traffic distortions in the outgoing cell streams, particularly when multiple high-σ connections are active simultaneously, as shown in Fig. 9(b). In contrast, the fair arbitration scheme effectively interleaves backlogged connections in a round-robin fashion, resulting in virtually no inflation in σ_{out}, even under heavy load. Since this experiment assigns the same ρ value to all connections, the weighted and unweighted fair architectures have identical performance.

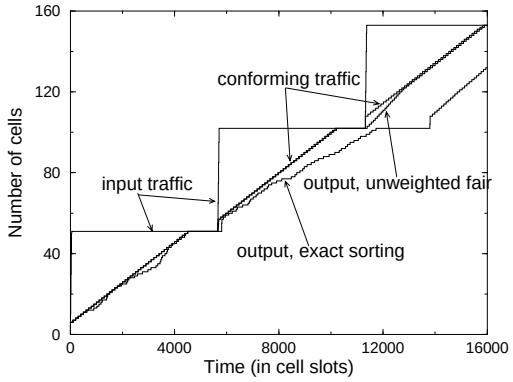
C. Mixing Different Bandwidth Parameters

Although the unweighted fair architecture guarantees a minimum bandwidth to each backlogged connection, on a small time scale, round-robin arbitration does not provide fair service when connections have different bandwidth parameters. To demonstrate this effect, Fig. 10 shows the performance of high-rate connections in the presence of a large number of low-rate connections, with a total link utilization of 79%:

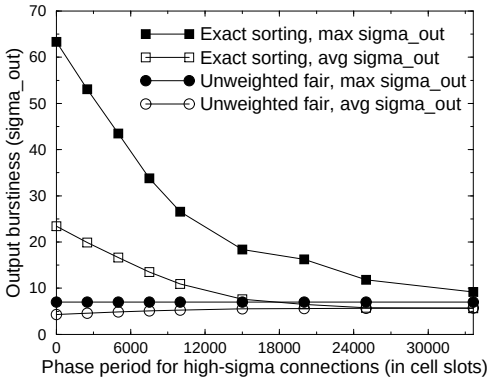
	Arrival		Shaping	
N	σ_{in}	ρ_{in}	σ	ρ
320	50	0.0022	25	0.0025
10	50	0.0090	5	0.0100

The experiment varies the phase of the 320 low-rate connections from 0 to their on/off period (22644), while the high-rate connections have uniform random start times during their on/off period. Fig. 10(a) plots the average output burstiness of the ten high-rate connections, while Fig. 10(b) plots the average delay in servicing conforming cells.

Although all three shaper architectures perform well under low levels of congestion, unweighted fair arbitration significantly inflates the burstiness of the high-rate connections during periods of heavy load, as shown in Fig. 10(a); the low-rate connections have $\sigma_{out} \approx \sigma$ throughout the experiment. Exact sorting also introduces traffic distortions, since the shaper services connections with different burstiness parameters. The high-rate connections also experience high cell shaping delays under these two architectures, as shown in Fig. 10(b); compared to exact sorting, the unweighted fair scheme has lower average delay by providing a minimum bandwidth to each high-rate connection. In contrast, weighted fair arbitration successfully preserves the leaky-bucket descriptors and limits cell shaping delay, even under extremely heavy congestion, by guaranteeing sufficient bandwidth to each connection on a small time scale. All three configurations have large shaping delays under small values of x, since all 320 low-rate connections

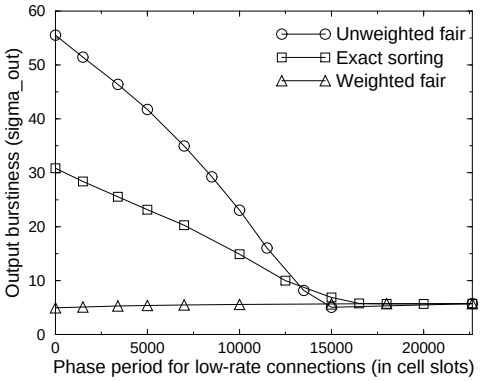


(a) Cell service (under phase of 11211)

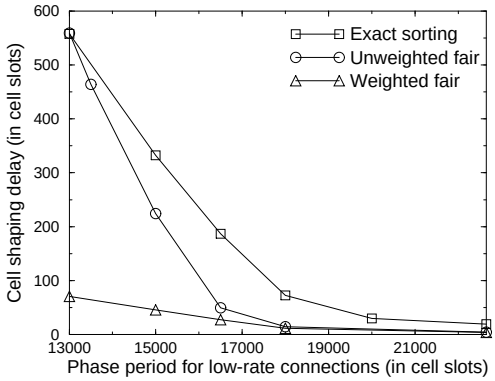


(b) Output burstiness (σ_{out} and $\overline{\sigma}_{out}$)

Fig. 9. **Mixing Different Burstiness Parameters:** This experiment compares the fair leaky-bucket shaper to an architecture that transmits cells in order of increasing conformance times. The plots highlight the performance of ten low-σ connections, which are mixed with eighty high-σ connections with the same ρ value. Under exact sorting, bursts of cells from these high-σ connections can temporarily deny service to the low-σ traffic, introducing traffic distortions and shaping delay.



(a) Average output burstiness $\overline{\sigma}_{out}$



(b) Average shaping delay

Fig. 10. **Mixing Different Rate Parameters:** This experiment compares the weighted fair traffic shaper to exact sorting and unweighted (round robin) arbitration amongst connections with conforming cells. The plots highlight the performance of ten high-rate connections, which are mixed with 320 bursty, low-rate connections. The weighted fair arbitration scheme performs well, even under heavy congestion, by guaranteeing sufficient bandwidth to the high-rate traffic on a small time scale.

have cells with nearly the same conformance times; hierarchical arbitration avoids this interference by segregating high-rate traffic into a separate group that receives its minimum bandwidth on a smaller time scale.

	Arrival		Shaping	
N	σ_{in}	ρ_{in}	σ	ρ
5	10	0.0909	0	0.1000
10	50	0.0094	25	0.0100
70	40	0.0048	20	0.0050

D. Hierarchical Arbitration

Although the weighted fair architecture performs well under diverse leaky-bucket parameters, handling a wide range of connection rates requires a large number of sorting bins. To quantify this cost-performance trade-off, Fig. 11 compares four different configurations of the proposed shaper, serving a mixture of connections that employ weighted fair arbitration, with a total link utilization of nearly 90%:

The experiment varies the phase period of the 70 low-rate connections from 0 to their on/off period (8528) to study the effects of congestion in the shaper; the other 15 connections have uniform random start times during their on/off periods.

The low-rate connections dictate the required range of the sorting bins, since a head-of-line cell can have a c value up to 200 time slots into the future. With coarse-grain sorting, the shaper can limit the number of bins:

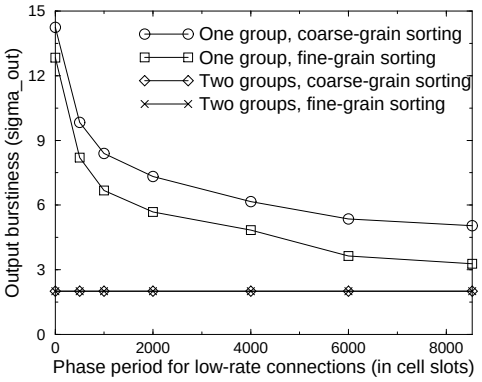


Fig. 11. **Hierarchical Fair Arbitration:** This experiment compares four configurations of the fair leaky-bucket traffic shaper handling a diverse mixture of connection burstiness and bandwidth parameters. The graph highlights the performance of five high-rate, constant-bit-rate connections, which are mixed with eighty low-rate, bursty connections.

Symbol	Configuration
◯	Single group with 201 bins of grain 1
□	Single group with 10 bins of grain 21
◇	High-rate group with 11 bins of grain 1; low-rate group with 201 bins of grain 1
×	High-rate group with 5 bins of grain 3; low-rate group with 5 bins of grain 41

at the expense of the burstiness parameters for the high-bandwidth traffic, as shown by comparing the top two curves in Fig. 11. To highlight the differences in the four configurations, the graph omits the "exact sorting" configuration, which performs dramatically worse, particularly for small values of x. For example, when $x = 4000$, exact sorting has an output burstiness of $\overline{\sigma}_{out} = 60$, in contrast to values of less than 8 for each of the configurations in Fig. 11. The low-rate connections do not experience significant traffic distortions under any of the shaper architectures. By dividing connections into two groups, the hierarchical architecture can reduce the number of sorting bins without distorting the high-bandwidth traffic, even under heavy congestion. In this configuration, one group services the 5 high-rate connections, while the second group handles the remaining traffic. Even with just 10 sorting bins, the coarse-grain hierarchical architecture has small $\overline{\sigma}_{out}$ values, since each group tailors its bin granularity to the connection bandwidth parameters.

Interestingly, the two hierarchical architectures even outperform the more expensive, fine-grain sorting scheme. This occurs because, even with weighted fair arbitration, a group can have multiple cells, from different connections, within a small range of sorting bins. Hence, when all connections share a single group, a high-rate connection may wait behind a long FIFO of low-bandwidth traffic before receiving service. 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. Such fair arbitration permits the hierarchical architecture to mix bursty, variable-bit-rate traffic with constant-bit-rate connections, without distorting the leaky-bucket parameters of the outgoing cell streams.

VI. Conclusion

Modern ATM switches require efficient traffic-shaping and link-scheduling hardware that can service a large number of connections with a wide range of bandwidth and burstiness parameters. High-speed links, coupled with small packet/cell sizes, require efficient architectures that can handle cell arrivals and departures every few microseconds, or faster. To reduce implementation complexity and improve performance, switches can incorporate per-connection queueing, approximate sorting algorithms, and hierarchical arbitration policies. With careful selection of sorting and arbitration schemes, these designs can also limit the traffic distortions that arise when multiple conforming cells await service. This is particularly important for shaping at the egress of a network, where the incoming traffic rate may temporarily exceed the capacity of the outgoing link, and in large-scale switches with multiple ports.

The paper addresses both implementation and performance scalability by presenting a logical progression of realizable shaper architectures. As discussed in Section II, most existing shapers include a sorting mechanism for ranking cells by their conformance times. Per-connection queueing has the potential to reduce shaper complexity by limiting the range of conformance times in the sorting logic. However, under exact sorting, per-connection queueing introduces challenges in handling a backlog of conforming traffic and avoiding per-cell timestamps. In addition, in times of congestion, exact sorting can unduly penalize connections with low σ values. The architecture in Fig. 5 reduces implementation complexity by placing all conforming head-of-line cells in a single FIFO queue, instead of ranking these cells by conformance time. Under congestion, this effectively provides round robin service to connections with conforming cells.

In contrast to exact sorting, round robin service enables the shaper to mix connections with different *burstiness* parameters without distorting the leaky-bucket descriptors of the outgoing cell streams. By allowing high-rate connections to insert *multiple* cells into the sorting unit, the shaper can also handle connections with a mix of different *bandwidth* parameters. This design effectively relegates part of the link-scheduling function into the shaping logic by implementing weighted round robin arbitration amongst connections with conforming cells. To efficiently handle a wider range of rate parameters, Fig. 8 groups connections based on their bandwidth requirements. Then, a second scheduler divides link bandwidth fairly among a small number of *groups*, avoiding the overhead of arbitrating between a large number of *connections* in the second stage. Heuristic group arbitration schemes, based on dynamic weights, can provide a fairer division of excess link bandwidth between individual backlogged connections.

This integrated approach to traffic shaping and link

scheduling has general utility in designing switch architectures that reshape traffic at each link, to reduce delay jitter and downstream buffer requirements. Ultimately, effective traffic shaping and link scheduling require a careful balance between implementation complexity and accuracy in approximating an idealized scheme. In contrast to the conceptual model in Fig. 1, the proposed architecture avoids both the replication of complex sorting logic and the expensive movement of cells between two separate priority queues. As future work, we are further analyzing the properties of the integrated architecture, through additional simulation experiments with more realistic traffic patterns. Finally, we are considering extensions to the hierarchical traffic-shaping and link-scheduling framework, including arbitration schemes that differentiate between ATM traffic classes, as well as support for connections that do not require reshaping at every switch.

Appendix

I. Avoiding Per-Cell Timestamps

In leaky-bucket traffic shaping, a cell's conformance time c depends on the status of the connection's token bucket *on cell arrival*. This suggests that a shaper must store the c value of each buffered cell to schedule access to the outgoing link. The shapers in Fig. 4 do not require such timestamps, since each arriving cell immediately enters a sorting bin. However, if the shaper employs per-connection queueing, incoming cells often wait in a connection FIFO before proceeding to the sorting unit. As a result, the shaper must have a mechanism for determining where to insert a new head-of-line cell in the sorting unit. For an exact sorting scheme, this requires the shaper to determine the c value of each head-of-line cell. In contrast, the approximate architecture in Fig. 5 places all *conforming* head-of-line cells in a single FIFO queue, obviating the need to recall their precise c values; hence, the shaper only needs to reconstruct c for *non-conforming* head-of-line cells.

Since a connection can store and consume credit tokens, the shaper should track the status of the token bucket over time. When a connection's token bucket is empty, arriving cells are non-conforming and have c values that are spaced $1/\rho$ apart, as in a peak-rate shaper. The shaper can reconstruct these cell conformance times by maintaining a small amount of per-connection state, as shown in Fig. 12. When a connection has exhausted its token bucket, the next arriving cell triggers the algorithm to save the state X_b of this "boundary cell," as shown in Fig. 13; this enables the shaper to reconstruct the c values for subsequent non-conforming arrivals. Whenever a cell arrives already conforming at time t, any earlier cells in that connection must also have $c \leq t$. Hence, when a conforming cell arrives, the algorithm *resets* the boundary position to null; this ensures that a connection has at most one boundary cell, even if the shaper falls behind in servicing conforming traffic.

Using the algorithm in Fig. 12 under per-connection queueing, the shaper can respond quickly and efficiently to dynamic changes in connection bandwidth parameters,

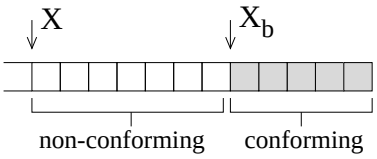


Fig. 13. **Boundary Between Conforming/Non-Conforming Cells:** Instead of storing the conformance time c of each cell, the shaper maintains the state of the leaky-bucket controller at two points for each connection. Variable X represents the state at the tail of the connection, as in Fig. 3, and X_b represents the state at the boundary between conforming and non-conforming cells.

based on feedback from the network or renegotiation from the source. The shaper can *implicitly* adjust conformance times by always applying the current value of ρ when computing c and updating X_b. The current leaky-bucket state X depends on X_b and the number of non-conforming cells in the connection FIFO; as a result, the shaper can adjust the leaky-bucket state

$$X \leftarrow X_b + (X - X_b)\rho^{old}/\rho^{new}$$

to capture the effects of the rate change on the connection token bucket. This applies the *new* rate ρ^{new} to any *non-conforming* cells that have not already entered the sorting bins, instead of waiting for the connection's backlog to clear the shaper. Otherwise, when $\rho^{new} < \rho^{old}$, the shaper would reward the connection by assigning smaller conformance times to these early arrivals; cells that were already conforming under rate ρ^{old} remain conforming under the new rate, since the burstiness parameter σ does not change.

II. Scheduling Concatenation Operations

Although hierarchical arbitration reduces the *number* of sorting bins, the shaper must periodically move cells from the sorting bins to the transmission FIFO in each group. As shown in the bottom of Fig. 6, this background operation requires only a simple concatenation of two linked lists, without initiating any data transfer on the link. In a scheduler with n groups, at most n such operations can occur in the same time unit t (i.e., when every group i has $t \bmod g_i = g_i - 1$ simultaneously). To further reduce this overhead, particularly in a software implementation of the shaper, the architecture can tighly bound the number of groups that require a concatenation operation in the same time slot. This optimization is possible because each group i only requires a concatenation operation once every g_i time units. The shaper can limit the number of sorting bins that become conforming simultaneously by *staggering* the conformance times of each group's bins, allowing the groups to share a single controller for appending the transmission FIFOs.

The shaper can handle each of the n groups by constructing a periodic schedule that performs a concatenation operation for group i once for every m operations for group $i+1$; this can be cast as a special case of the pinwheel problem [33]. For example, $m = 2$ and $n = 3$ result in an 8-slot

Cell Arrival

```
X = max{X + 1/ρ, t};
if (X ≤ t + σ/ρ) { // Conforming cell
      boundary = ∅;
      if (connection is idle)
            cell has c ≤ t;
      else // Connection is backlogged
            can reconstruct c later;
}
else { // Non-conforming cell
      if (boundary == ∅) // New boundary
            boundary = current_cell;
            X_b = X − 1/ρ;
            if (connection is idle)
                  cell has c = X − σ/ρ;
            else // Connection is backlogged
                  can reconstruct c later;
}
```

Cell Departure

```
transmit current_cell;
if (connection is idle)
      boundary = ∅;
else { // Connection is still backlogged
      if (boundary != current_cell)
            next_cell has c ≤ t;
      else{ // Update for boundary cell
            boundary = next_cell;
            X_b = X_b + 1/ρ;
            if (X_b − σ/ρ ≤ t)
                  next_cell has c ≤ t;
            else // Reconstruct conformance time
                  next_cell has c = X_b − σ/ρ;
      }
}
```

Fig. 12. **Avoiding Per-Cell Timestamps:** The proposed shaping algorithm can employ per-connection queueing without requiring per-cell timestamps. Instead, the shaper can reconstruct conformance times based on per-connection state. Initially, $X = -1/\rho$ and `boundary`$= \emptyset$.

schedule:

2	1	2	0	2	1	2	–
0	1	2	3	4	5	6	7

For an efficient hardware implementation, the shaper can include a 3-bit counter, selecting a group based on the bit string (e.g., xx0 for group 2, x01 for group 1, and 011 for group 0); in particular, the shaper visits group i whenever the counter's least significant 0-bit is in position i. For general values of m and n, the schedule has m^n entries, represented by an n-digit, base-m number; group i is scheduled every m^{n-i} slots, when the counter has value

$$\underbrace{\text{x}\ldots\text{x}}_{i}\underbrace{01\ldots1}_{n-i}$$

in base m. The shaper must sequence through this periodic schedule quickly enough to visit each group i every g_i time units, requiring the schedule to operate at rate $r = m^{n-i}/g_i$,

$$r = m^{n-i}(um^{i+1}\rho_{\min}) = um^{n+1}\rho_{\min} = um\rho_{\max}.$$

For reasonable values of m, u, and $\rho_{\max}$, the shaper can perform operations on conforming bins at the link's cell transmission rate, or even slower. Otherwise, the shaper can have dedicated logic to perform concatentation operations in each group.

III. Group Arbitration with Dynamic Weights

Most rate-based scheduling algorithms arbitrate between busy sources based on virtual finishing times F_i, $i = 0, 1, \ldots, n-1$; the link transmits a cell from the backlogged group with the minimum F_i value in each transmission slot. To demonstrate the use of dynamic group weights, Fig. 14 shows a variation of the *self-clocked fair queueing* [17–20] algorithm. Self-clocked fair queueing employs a relatively simple recurrence for computing F_i values and closely approximates the more complicated weighted fair queueing algorithm, particularly for small values of n [19, 20]. To reflect the bandwidth consumed by each group, the algorithm increments F_i by $1/\phi_i$ after each transmission from group i. To arbitrate fairly based on weights ϕ_i, the arbiter assigns $F_i = F^{serv} + 1/\phi_i$, whenever group i becomes backlogged, where F^{serv} is the virtual finishing time of the group currently in service; this ensures that a previously idle group cannot claim credit for past bandwidth.

The switch can employ a small tree of comparators to efficiently identify the backlogged group with the minimum virtual finishing time; the server handles a bounded range of group F values, permitting modulo arithmetic and registers that store values ranging from 0 to $1/\min_i\{\phi_i\}$. Since the algorithm in Fig. 14 permits the ϕ_i values to change over time, the arbiter may need to *adjust* F_i when connection α_i becomes busy (causing an increase in ϕ_i). This heuristic portion of the algorithm [26] maintains the value ϕ_i^{old}, the value of ϕ_i the last time F_i was assigned; ϕ_i^{old} is set to ϕ_i whenever F_i changes. 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. 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. Although this heuristic arbitration scheme can sometimes deviate from

Cell Arrival

```
cell arrives for connection α_i in group i;
if (connection α_i is currently idle) {
    φ_i = φ_i + ρ_{α_i}; // Update group weight
    if (group i is currently idle)
        F_i = F^serv + 1/φ_i; φ_i^old = φ_i;
    else { // Group is already backlogged
        // Might adjust F_i for higher rate
        if (F^serv ≤ F_i − 1/φ_i^old + 1/φ_i )
            F_i = F_i − 1/φ_i^old + 1/φ_i; φ_i^old = φ_i;
        else if (F_i > F^serv + 1/φ_i)
            F_i = F^serv + 1/φ_i; φ_i^old = φ_i;
        else
            // F_i and φ_i^old stay the same
    }
}
add new cell to group i;
```

Cell Departure

```
find group i with min F_i and non-empty TXFIFO;
transmit head cell from group i's TXFIFO;
F^serv = F_i;
if (connection α_i is now idle)
    φ_i = φ_i − ρ_{α_i};
else // Connection is still backlogged
    // No need to change group weight φ_i
if (group i is still backlogged)
    F_i = F^serv + 1/φ_i; φ_i^old = φ_i;
else // Group i is now idle
    // No need to assign F_i for an idle group
```

Conforming Bin

```
if (bin non-empty) and (TXFIFO empty)
    if (F_i < F^serv)
        F_i = F^serv + 1/φ_i; φ_i^old = φ_i;
else
    // No need to reset F_i
concatenate bin with group i's TXFIFO;
```

Fig. 14. **Dynamic Hierarchical Arbitration:** By allowing the group weights ϕ_i to change over time, the shaper can approximate connection-level fairness with a heuristic group arbitration scheme, based on self-clocked fair queueing. The weight ϕ_i, representing the aggregate rate of all backlogged connections in group i, is adjusted whenever a connection α_i becomes backlogged or idle. Initially, $\phi_i = \phi_i^{old} = 0$, $F_i = 0$, and $F^{serv} = 0$.

connection-level fairness [26], the shaping function ensures that each connection makes forward progress based on the leaky-bucket conformance times of its incoming cells; a group continues to compete for link access until it clears its temporary backlog of conforming cells. Ultimately, static weights ϕ_i can provide a minimum rate to each connection on a small time scale, while dynamic weights can provide a fairer division of excess link bandwidth on a coarser time scale. Both approaches provide an efficient, event-driven way to multiplex a large number of connections with diverse bandwidth requirements.

Acknowledgments

The authors would like to thank the anonymous referees for their thorough reviews and conscientious suggestions.

References

[1] Flavio Bonomi and Kerry W. Fendick, "The rate-based flow control framework for the available bit rate ATM service," *IEEE Network Magazine*, pp. 25–39, March/April 1995.

[2] Hui Zhang and Domenico Ferrari, "Rate-controlled service disciplines," *Journal of High Speed Networks*, vol. 3, no. 4, pp. 389–412, 1994.

[3] Hui Zhang, "Providing end-to-end performance guarantees using non-work-conserving disciplines," *Computer Communications*, vol. 18, no. 10, pp. 769–781, October 1995.

[4] Leonidas Georgiadis, Roch Guerin, Vinod Peris, and Kumar N. Sivarajan, "Efficient network QoS provisioning based on per node traffic shaping," *IEEE/ACM Trans. Networking*, vol. 4, no. 4, pp. 482–501, August 1996.

[5] Jennifer Rexford, John Hall, and Kang G. Shin, "A router architecture for real-time point-to-point networks," in *Proc. Inter. Symposium on Computer Architecture*, May 1996, pp. 237–246.

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

[7] H. Jonathan Chao and Necdet Uzun, "A VLSI sequencer chip for ATM traffic shaper and queue manager," *IEEE J. of Solid-State Circuits*, vol. 27, no. 11, pp. 1634–1643, November 1992.

[8] Pierre E. Boyer, Fabrice M. Guillemin, Michel J. Servel, and Jean-Pierre Coudreuse, "Spacing cells protects and enhances utilization of ATM network links," *IEEE Network Magazine*, pp. 38–49, September 1992.

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

[10] Tim Moors, Nathan Clarke, and Guven Mercankosk, "Implementing traffic shaping," in *Proc. Conference on Local Computer Networks*, October 1994, pp. 307–314.

[11] Guven Mercankosk, Tim Moors, and Antonio Cantoni, "Multiplexing spacer outputs on cell emissions," in *Proc. IEEE INFOCOM*, April 1995, pp. 49–55.

[12] Jonathan Turner, "New directions in communications, or which way to the information age?," *IEEE Communication Magazine*, vol. 24, no. 10, pp. 8–15, October 1986.

[13] ATM Forum, "Traffic management specification, version 4.0," ATM Forum/95-0013R10, February 1996.

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

[15] Abhay K. Parekh and Robert G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single node case," *IEEE/ACM Trans. Networking*, vol. 1, no. 3, pp. 344–357, June 1993.

[16] Abhay K. Parekh and Robert G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The multiple node case," *IEEE/ACM Trans. Networking*, vol. 2, no. 2, pp. 137–150, April 1994.

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

[18] Jim W. Roberts, "Virtual spacing for flexible traffic control," *International Journal of Communication Systems*, vol. 7, no. 7, pp. 307–318, October–December 1994.

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

[20] S. Jamaloddin Golestani, "Network delay analysis of a class of fair queueing algorithms," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 6, pp. 1057–1070, August 1995.

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

[22] Jennifer Rexford, Flavio Bonomi, Albert Greenberg, and Albert Wong, "A scalable architecture for fair leaky-bucket shaping," in *Proc. IEEE INFOCOM*, April 1997.

[23] Jon C. R. Bennett and Hui Zhang, "Hierarchical packet fair queueing algorithms," in *Proc. ACM SIGCOMM*, August 1996, pp. 143–156.

[24] Dimitrios Stiliadis and Anujan Varma, "A general methodology for designing efficient traffic scheduling and shaping algorithms," in *Proc. IEEE INFOCOM*, April 1997.

[25] Subhash Suri, George Varghese, and Girish Chandranmenon, "Leap forward virtual clock: A new fair queuing scheme with guaranteed delays and throughput fairness," in *Proc. IEEE INFOCOM*, April 1997.

[26] Jennifer Rexford, Albert Greenberg, and Flavio Bonomi, "Hardware-efficient fair queueing architectures for high-speed networks," in *Proc. IEEE INFOCOM*, March 1996, pp. 638–646.

[27] H. Jonathan Chao, "A novel architecture for queue management in the ATM network," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 7, pp. 1110–1118, September 1991.

[28] Randy Brown, "Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem," *Communications of the ACM*, vol. 31, no. 10, pp. 1220–1227, October 1988.

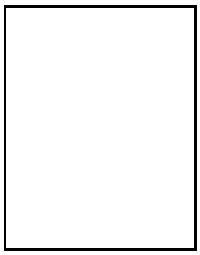
[29] Fouad A. Tobagi, "Fast packet switch architectures for broadband integrated services digital networks," *Proceedings of the IEEE*, vol. 78, no. 1, pp. 133–167, January 1990.

[30] H. Jonathan Chao and Donald E. Smith, "A shared-memory virtual channel queue for ATM broadband terminal adaptors," *International Journal of Digital and Analog Communication Systems*, vol. 5, no. 2, pp. 29–37, January–March 1992.

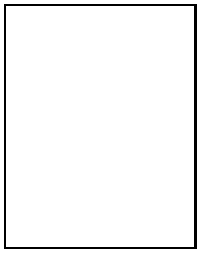
[31] Scott Shenker, David D. Clark, and Lixia Zhang, "A scheduling service model and a scheduling architecture for an integrated services packet network," Working paper, Xerox PARC, August 1993.

[32] Sally Floyd and Van Jacobson, "Link-sharing and resource management models for packet networks," *IEEE/ACM Trans. Networking*, vol. 3, no. 4, pp. 365–386, August 1995.

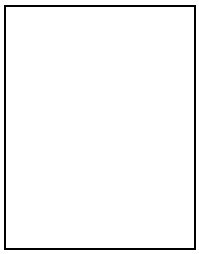
[33] Robert Holte, Al Mok, Igor Tulchinsky, and Donald Varvel, "The Pinwheel: A real-time scheduling problem," in *Proc. Hawaii International Conference on System Sciences*, January 1989, pp. 693–702.



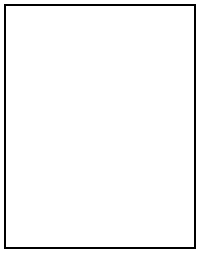
Jennifer Rexford received a B.S.E. degree in electrical engineering from Princeton University in 1991, and M.S. and Ph.D. degrees in computer science and engineering from the University of Michigan in Ann Arbor, in 1993 and 1996, respectively. Since 1996, she has been in the Network Mathematics Research department at AT&T Labs Research in New Jersey. Her research interests include packet scheduling, routing/signalling protocols, and performance evaluation, with an emphasis on efficient support for quality-of-service guarantees. Her e-mail address is jrex@research.att.com.



Flavio Bonomi received a Laurea in electrical engineering from Pavia University, Italy, in 1978, and M.S. and Ph.D. degrees in electrical engineering from Cornell University, Ithaca, New York, in 1981 and 1985, respectively. From 1985 to 1995 he was with AT&T Bell Laboratories, in Holmdel and later Red Bank, New Jersey. While at AT&T he was involved in development and research on ATM broadband systems, with particular focus in switch architecture design, performance analysis, traffic management and network engineering. Since 1995 he has been the Principal Architect at ZeitNet/Cabletron in Santa Clara, California, with responsibilities in the definition of broadband host adapter and switching products. His e-mail address is flavio@zeitnet.com.



Albert Greenberg received a B.A. in mathematics at Dartmouth College in 1978, and a Ph.D. in computer science at the University of Washington in 1983. From 1983 to 1995 he was with AT&T Bell Laboratories in Murray Hill, New Jersey. Since 1995, he has been head of the Network Mathematics Research department at AT&T Labs Research. His research focuses on the measurement, modeling, and analysis of networks, including work on parallel computing and techniques for fast simulation. His e-mail address is albert@research.att.com.



Albert Wong was born in Hong Kong on November 28, 1959. He received the S.B., S.M., E.E., and Ph.D. degrees from the Massachusetts Institute of Technology in 1982, 1984, 1984, and 1988, respectively, all in electrical engineering. He joined Bell Laboratories, New Jersey, in 1988 and became part of Lucent Technologies in 1996. From 1988 to 1992, he worked on photonic switching and high-speed networking research. Since 1992, he has been working on the development of ATM switching systems and is currently technical manager of an ATM traffic and performance group. Dr. Wong has also been an adjunct faculty member of Polytechnic University of New York and of Rutgers University. He received Bell Laboratories' Distinguished Member of Technical Staff award in 1993. His e-mail address is albertwong@lucent.com.