BSFQ: Bin Sort Fair Queueing

Shun Y. Cheung and Corneliu S. Pencea
Department of Mathematics and Computer Science,
Emory University, Atlanta, Georgia 30322

***Abstract*— Existing packet schedulers that provide fair sharing of an output link can be divided into two classes: sorted priority and frame-based. Sorted priority methods provide excellent approximation for Weighted Fair Queueing (WFQ) while frame-based methods are more computationally efficient. We present a new packet scheduling algorithm called Bin Sort Fair Queueing (BSFQ) that combines the strengths of both type of schedulers. As a result, BSFQ is highly scalable and can provide very good approximation for WFQ. We prove that BSFQ can provide end-to-end delay and fairness guarantees to conformant flows. BSFQ also has a built-in buffer management function that can protect packets of conformant flows from non-conformant traffic. The performance of BSFQ and its ability to detect non-conformant flows are studied using simulations and compared to those of the Deficit Round Robin method.**

***Index Terms*— Quality of Service, Delay Guarantee, Fairness Guarantee, WFQ, BSFQ**

I. Introduction

THE Internet has enjoyed tremendous growth in the recent years and the traffic on the Internet is more diverse than ever – ranging from traditional email to real time audio and video applications. Many novel applications are well-served if the network can provide a certain degree of performance. For instance, the performance of audio applications over the Internet would be greatly improved if the network would be able to provide delay and/or rate guarantees.

Quality of service provisioning to flows is realized by insulating the flows from one another. The ideal packet scheduling method to ensure fairness is the Fluid Fair Queueing (FFQ) method [2]. In this hypothetical packet service discipline, data of active flows are transmitted one bit at a time in a round robin fashion. This service discipline is not practical as packets must be transmitted in their entirety. The Weighted Fair Queueing (WFQ) [2] and Worst-Case Fair WFQ (WF^2Q) [3] are packet schedulers that mimic FFQ as closely as possible. These methods are computationally expensive and other more efficient scheduling techniques have been developed that *approximate* the behavior of WFQ. The methods can be categorized as *sorted-priority* or *frame-based* [5].

In sorted-priority schemes, each packet is assigned a priority value and packets are transmitted in increasing order of their priority. To approximate the transmission order in the idealized FFQ system, the priority value assigned to a packet is some function of its departure time in the FFQ system. Examples of sorted-priority schemes are Virtual Clock (VC) [6], Self-Clock Fair Queueing (SCFQ) [7] and Leap-Forward Virtual Clock (LFVC) [8]. In the frame-based approach, time is divided into frames and packets are entered into a frame without exceeding a maximum. An example of a frame-based scheme is the Deficit Round Robin (DRR) method [9]. Frame-based methods are very scalable as the packet processing operations have constant time ($O(1)$) complexity, in contrast to the sorted-priority schemes that require a sorting operation to insert a new packet.

In this paper, we present the new Bin Sort Fair Queueing (BSFQ) packet scheduling method that is a frame-based method and uses priority assignments in sorted-priority schemes to determine the packets that are transmitted in each round. In BSFQ, the *virtual time* is divided into slices of equal length called *bins*. As in sorted-priority schemes, each arriving packet is stamped with a priority value which represents its *virtual departure time*. The packet is then stored in the bin that corresponds to the time slice containing the virtual departure time of that packet. The packets stored in the same bin are queued in FIFO order for efficiency reasons (but other more sophisticated scheduling method can optionally be used). As a result, BSFQ combines the strengths of both types of scheduling methods: it is highly efficient and provides better approximation for WFQ than existing frame-based schemes.

The paper is organized as follows. Section II presents an overview of the related work. We present the BSFQ method in Section III and show that the BSFQ method can provide end-to-end delay and fairness guarantees. Section IV presents a simulation study of its performance. The paper is concluded in Section V.

II. Related Work

The general processor sharing (GPS) [1] server is the ideally fair service discipline that can allocate a predefined share of service capacity to each client or network flow. If there is a single client obtaining service from a GPS server, the client will be served at the rate of the server. However, when two different clients are present, both will be serviced simultaneously but each will receive half the service rate. The service rates received by each client can also be weighted differently. The GPS service method is also called Fluid Fair Queueing (FFQ) in [2].

FFQ provides perfect fairness to network flows. However, FFQ is not a practical transmission procedure because packets must be transmitted as an atomic unit.

We will briefly review the basic concepts to analyze the fairness of scheduling algorithms. A flow is *backlogged* if it has some packets in the output buffer. A packet scheduling method is *fair* if the difference in the normalized service provided to any two flows that are continuously backlogged over any interval $[t_1, t_2]$ is bounded by some constant [7]. The normalized service $w_f(t_1, t_2)$ received by flow f in $[t_1, t_2]$ is defined as $\frac{W_f(t_1,t_2)}{r_f}$, where $W_f(t_1, t_2)$ is the amount of data of flow f transmitted in $[t_1, t_2]$ and r_f is the reserved data rate of f. $W_f(t_1, t_2)$ includes any part of packets from f transmitted in $[t_1, t_2]$. If a scheduling discipline is fair then there is a constant ϵ such that $|w_f(t_1, t_2) - w_g(t_1, t_2)| \leq \epsilon$, for any two flows f and g that are backlogged during the interval $[t_1, t_2]$. The constant ϵ is independent of the length of the interval $[t_1, t_2]$. In the ideally fair FFQ method, we have that for any times $t_1 < t_2$, $w_f(t_1, t_2) = w_g(t_1, t_2)$ for any two flows f and g that are backlogged during $[t_1, t_2]$. Because packets must be transmitted as a unit, packet schedulers will have $|w_f(t_1, t_2) - w_g(t_1, t_2)| > 0$.

Packet schedulers that approximate the FFQ discipline are called Packet-by-packet Fair Queueing (PFQ) [7]. The WFQ [2] and WF^{2}Q [3] packet schedulers provide the most accurate approximations of FFQ. These methods first compute the "virtual finish time" of a packet using the FFQ scheduler as reference and then transmits the packets in ascending finish time values. The WFQ scheduler does so for every packet in the output buffer while the WF^{2}Q scheduler only considers those packets that would have started receiving service in the corresponding FFQ system.

More computationally efficient (but less accurate) packet scheduling schemes have been developed. The Virtual Clock method [6] is one of the earliest methods proposed to insulate network flows. The VC method assigns the j^{th} packet p_f^j of flow f with the virtual time stamp $vts_{VC}(p_f^j) = \max(A(p_f^j), vts_{VC}(p_f^{j-1})) + \frac{\ell_f^j}{r_f}$, for $j > 0$, where $A(p_f^j)$ is the arrival time of packet p_f^j and ℓ_f^j is the length of p_f^j. The virtual time stamp $vts_{VC}(p_f^0)$ is set to zero. Xie and Lam showed in [11] that if the sum of the rate of all flows sharing a link does not exceed the link capacity, then the departure time $L(p_f^j)$ of packet p_f^j is bounded by $L(p_f^j) \leq vts_{VC}(p_f^j) + \frac{\ell_f^{max}}{R}$ where ℓ_f^{max} is the length of the largest packet of flow f and R is the data rate of the output link. Although VC provides a delay guarantee to flows, it does not provide any fairness guarantee. Packets from a flow that has been idle for a prolonged period of time will be assigned smaller virtual time stamp values and can receive — for some period of time — a larger than reserved share of service.

The Leap Forward Virtual Clock [8] method solves the fairness problem in VC by temporarily moving oversubscribed flows into a low priority holding area. Only flows in the high priority area will receive service. A flow f is oversubscribed if the difference between the virtual time stamps of the current packet of f and the system time exceeds a certain threshold. In the case when all flows are oversubscribed, the system clock is advanced forward to allow some flows to be moved back into the high priority area.

The Self-Clocked Fair Queueing method [7] uses an internal (virtual) system clock $\tau(t)$ to compute time stamps for packets. The system clock $\tau(t)$ is equal to the virtual time stamp of the packet that is being serviced at time t. The j^{th} packet p_f^j of flow f will receive the virtual time stamp $vts_{SCFQ}(p_f^j) = \max(\tau(A(p_f^j), vts_{SCFQ}(p_f^{j-1})) + \frac{\ell_f^j}{r_f}, j > 0$. The value $vts_{SCFQ}(p_f^0)$ is set to zero and the packets are transmitted in the ascending order of their virtual time stamp values. SCFQ can provide both an end-to-end delay guarantee [12] and a fairness guarantee [7] to conformant flows.

The VC, LFVC and SCFQ schedulers are priority-based schemes where packets are transmitted in ascending virtual time stamp values. Priority-based schemes use some sort procedure to maintain a priority queue and have non-constant per packet run time complexities. In contrast, frame-based methods transmit packets in rounds. They do not use sort operations and have constant per packet processing time. The disadvantage of frame-based methods is the fact that the delay guarantee they provide have a larger bound than sorted-priority methods.

The Deficit Round Robin (DDR) method [9] organizes packets of flows in separate queues and assigns a quantum size to each flow. Each flow has a "deficit counter" that measures the current unused portion of the allocated bandwidth. Packets of backlogged flows are transmitted in rounds and in each round, each backlogged flow can transmit up to an amount of data equal to the sum of its quantum and deficit counter. The unused portion of this amount is carried over to the next round as the deficit counter value.

The Uniform Round Robin (URR) method [10] is a cell-based method that can reduce the jitter of the DRR method by spacing cell transmissions uniformly over a round. For example, when four flows A, B, C and D with with quantum value $q_A = 3$ cells and $q_B = q_C = q_D = 1$ cell are backlogged, a possible transmission order per round in DRR is (A, A, A, B, C, D). In contrast, the order of transmission in URR is (A, B, A, C, A, D). The transmission ordering in URR must be recomputed when a new flow is added or an existing flow is deleted from the schedule and this recomputation algorithm is $O(s)$ where s is the number of slots in one round. After the ordering is determined, the per packet processing time is constant.

The BSFQ method presented in this paper is a frame-based method and uses virtual time stamps to determine the scheduling order. The virtual time space is divided into equal intervals or bins. Packets are assigned virtual time stamps and inserted into their corresponding bins. The queueing order within a bin

is FIFO. The BSFQ method has constant run time complexity for connection establishment and packet processing. A simulation study will show that BSFQ can provide a better approximation for WFQ than DRR using similar operational parameters. WFQ also has a built-in buffer management component and its effectiveness to identify packets from non-conformant flows is illustrated using another simulation experiment.

III. Bin Sort Fair Queueing

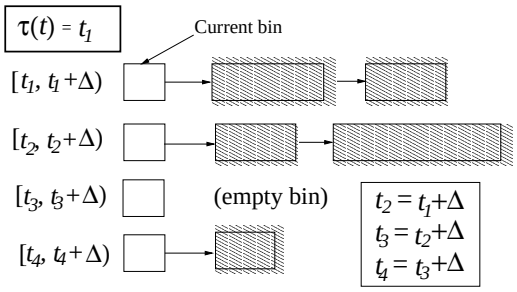

Fig. 1. Bin Sort Fair Queueing

The BSFQ method is defined for output buffer switches. Figure 1 shows the logical organization used in BSFQ. The output buffer is organized into N bins and N is a system design parameter. N must be set to a value that is equal to or greater than a certain threshold to allow BSFQ to use all available buffers. This threshold value will be derived later in this section. Each bin is implicitly labeled with a *virtual time interval* and each interval has length Δ. The parameter Δ is another system design parameter of BSFQ and its value has a significant impact on BSFQ's performance. The intervals of the bins are disjoint and their union spans a continuous range of the virtual time space.

The bins are ordered according to their virtual time intervals and are serviced in that ordering. The *current bin* is initially equal to the bin with virtual time interval [0, Δ) and only packets from the current bin are transmitted. When all packets in the current bin are transmitted, the next bin becomes the current one and the old current bin is labeled with an interval that follows the last bin. Thus, there are always N bins in the system.

BSFQ maintains a virtual system clock $\tau(t)$ which is equal to the left end point of the virtual time interval of the current bin at time t. Thus, the current bin has the label $[\tau(t), \tau(t) + \Delta)$ and the i^{th} bin following the current bin has the label $[\tau(t) + i\Delta, \tau(t) + (i+1)\Delta)$. The virtual time clock in BSFQ is a step function — similar to the virtual clock in SCFQ — and $\tau(t)$ will be incremented by Δ whenever all packets in the current bin are transmitted. Notice that if a bin is empty when it becomes the current bin, then $\tau(t)$ is incremented by Δ without transmitting any packets. We will see later that when Δ is sufficiently small, the performance of BSFQ will approximate that of WFQ.

We denote the data rate of the output link by R and each flow f must negotiate a guaranteed rate r_f before starting its transmission. We assume that $\sum_{all f} r_f \leq R$ so that the output link is not oversubscribed. The j^{th} packet p_f^j of flow f is assigned with the virtual time stamp $vts(p_f^j)$ where:

$$vts(p_f^j) = \max(\tau(A(p_f^j)), vts(p_f^{j-1})) + \frac{\ell_f^j}{r_f},\ j > 0 \qquad (1)$$

where $\tau(A(p_f^j))$ is the system virtual time at time $t = A(p_f^j)$. We define $vts(p_f^0) = 0$. Arriving packets are stored in their corresponding bins in the FIFO order. The index i_f^j of the bin used to store packet p_f^j is equal to:

$$i_f^j = \left\lfloor \frac{vts(p_f^j) - \tau(A(p_f^j))}{\Delta} \right\rfloor,\quad j > 0 \qquad (2)$$

If $i_f^j = 0$ then p_f^j is stored in the current bin, and otherwise, if $i_f^j < N$, it is stored in the i_f^j-th bin following the current bin. If $i_f^j \geq N$, the packet is discarded. Furthermore, if p_f^j has been discarded by BSFQ, then the packet index (j) is not incremented and next arriving packet of flow f will have the same index as the discarded one. Hence, the BSFQ scheduler has a built-in buffer management component. A simulation study will show that the buffer management function in BSFQ can effectively protect packets of compliant flows from non-compliant ones.

The following examples show the packet scheduling operation in BSFQ and the effect of the parameter Δ on its performance. Consider two flows A and B with reserved rates $r_A = 3000$ bps and $r_B = 1000$ bps, respectively. Assume that a large number of packets from both flows arrive at the switch simultaneously at time 0, and each packet is 9000 bits in length. The WFQ scheduler will transmit the packets as: p_A^1, p_A^2, $\{p_A^3, p_B^1\}$, p_A^4, p_A^5, $\{p_A^6, p_B^2\}$, etc. The packets $\{p_A^3, p_B^1\}$, $\{p_A^6, p_B^2\}$, and so on, may be transmitted in either order by WFQ.

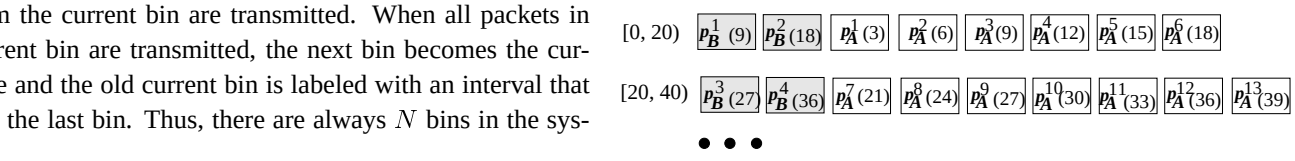

Fig. 2. BSFQ using $\Delta = 20$

We now consider the transmission order in BSFQ. The virtual time stamps of A's and B's packets assigned by BSFQ are: $p_A^1 = 3$, $p_A^2 = 6$, $p_A^3 = 9$, etc. and $p_B^1 = 9$, $p_B^2 = 18$, $p_B^3 = 27$, and so on. Figure 2 shows the queueing order in a BSFQ server for $\Delta = 20$. We have assumed that packets from B arrive just before those of A and due to the FIFO ordering, B's packets are queued before those of A if they are entered in the same bin. We can see from Figure 2 that the packet transmission order in BSFQ is $\mathbf{p_B^1}$, $\mathbf{p_B^2}$, p_A^1, p_A^2, p_A^3, p_A^4, p_A^5, p_A^6, $\mathbf{p_B^3}$, $\mathbf{p_B^4}$, p_A^7, p_A^8, p_A^9, p_A^{10}, p_A^{11}, p_A^{12}, p_A^{13}, and so on. BSFQ allocates for each flow its fair share of bandwidth as the transmission rate of flow A is three times that of flow B. However, the transmission order

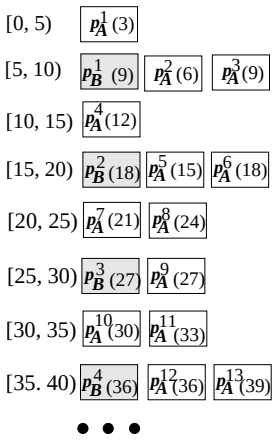

Fig. 3. BSFQ using $\Delta = 5$

differs significantly from the one in WFQ above. For instance, p_B^2 is transmitted before p_A^1, p_A^2, p_A^3, p_A^4 and p_A^5.

Consider the same arrival pattern in the BSFQ system that uses a smaller value of Δ. Figure 3 shows the queueing order for $\Delta = 5$. We have again assumed that all packets from B are queued before those of A when they are in the same bin. We can see from Figure 3 that the packet transmission order is now $p_A^1, \mathbf{p_B^1}, p_A^2, p_A^3, p_A^4, \mathbf{p_B^2}, p_A^5, p_A^6, p_A^7, p_A^8, \mathbf{p_B^3}, p_A^9, p_A^{10}, p_A^{11}, \mathbf{p_B^4}, p_A^{12}, p_A^{13}$, and so on. The ordering is identical to the one of a WFQ server except for p_B^1 which is transmitted before p_A^2 and p_B^2 that is serviced before p_A^5. We see from these two examples that Δ has a significant impact on BSFQ's performance.

BSFQ can also accommodate flows that do not make reservations. We define the residual rate r_{res} to be equal to $R - \sum_{\text{all flows } f} r_f$. Notice that r_{res} will change when new reservations are made or an existing reservation is released. Packets from any of the non-reservation flows are assigned a time stamp value using r_{res} as reserved rate and then scheduled in the same manner as other reservation flows.

In the remainder of this section, we will prove that BSFQ provides an end-to-end delay and a fairness guarantee.

A. Delay Guarantee

The work in [12] presents a class of schedulers that can provide an end-to-end delay guarantees for leaky bucket rate controlled sources. A packet scheduler is in class GR if the departure time $L(p_f^j)$ of packet p_f^j is bounded by

$$L(p_f^j) \leq GRC(p_f^j) + \beta$$

for some constant β. $GRC(p_f^j)$ for $j > 0$ is defined as:

$$GRC(p_f^j) = \max(A(p_f^j), GRC(p_f^{j-1})) + \frac{\ell_f^j}{r_f} \qquad (3)$$

and $GRC(p_f^0) = 0$. According to [12], the end-to-end delay d_f^j of packet p_f^j that traverses a path of K nodes, each using a GR-scheduler, is bounded by $d_f^j \leq GRC^1(p_f^j) - A^1(p_f^j) + (K-1)\max_{n=1}^j \frac{\ell_f^n}{r_f} + \sum_{n=1}^K \left(\beta^n + \delta^{n,n+1}\right)$, where $GRC^1(p_f^j)$ is the value of $GRC(p_f^j)$ at node 1, $A^1(p_f^j)$ is the arrival time of p_f^j at the first node of the path, β^n is the constant of node n on the path and $\delta^{n,n+1}$ is the propagation delay between nodes n and $n+1$, for $n = 1, \ldots, K$. Furthermore, if flow f is a leaky bucket compliant flow with parameters (σ_f,ρ_f), then $d_f^j \leq \frac{\sigma_f + (K-1)\max_{n=1}^j \frac{\ell_f^n}{r_f}}{r_f} + \sum_{n=1}^K \left(\beta^n + \delta^{n,n+1}\right)$.

We will show that BSFQ can provide an end-to-end delay guarantee by showing that BSFQ belongs to the class GR as defined in [12]. We first show that the number of bytes admitted into k consecutive bins is bounded and define the "size" of a bin.

Lemma 1: The number of bits of data D_k admitted into k consecutive bins is bounded by:

$$D_k \leq k\Delta R + \sum_{all\ f} \ell_f^{max}$$

Proof: Let us consider the packets of a particular flow f that are entered into the bins $i, i+1, \ldots, i+k-1$, for an arbitrary $i \geq 0$. Assume that the first and last packet of f entered into these bins are p_f^F and p_f^L, respectively. Bin j contains only packets with time stamps between $j\Delta$ and $(j+1)\Delta$, for $j = i, \ldots, i+k-1$. Therefore, we have that $vts(p_f^F) \geq i\Delta$ and $vts(p_f^L) < (i+k)\Delta$. From (1), we have:

$$\begin{aligned} vts(p_f^L) &\geq vts(p_f^{L-1}) + \frac{\ell_f^L}{r_f} \\ &\geq vts(p_f^{L-2}) + \frac{\ell_f^{L-1} + \ell_f^L}{r_f} \\ &\ldots \\ &\geq vts(p_f^F) + \frac{\ell_f^{F+1} + \ldots + \ell_f^L}{r_f} \end{aligned}$$

Therefore,

$$\begin{aligned} \ell_f^F + \ldots + \ell_f^L &\leq (vts(p_f^L) - vts(p_f^F))r_f + \ell_f^{max} \\ &\leq k\Delta r_f + \ell_f^{max} \end{aligned}$$

Hence, the number of bits of data of flow f entered into bins i through $i+k-1$ is at most $k\Delta r_f + \ell_f^{max}$. The maximum number of data from all flows that are entered in these bins is thus bounded by:

$$\begin{aligned} D_k &\leq k\Delta \sum_{all\ f} r_f + \sum_{all\ f} \ell_f^{max} \\ &\leq k\Delta R + \sum_{all\ f} \ell_f^{max} \end{aligned}$$

■

D_k can be considered as the "capacity" of k consecutive bins. Since $\lim_{k\to\infty} \frac{D_k}{k} = \Delta R$, we define the *size* of a bin λ_{bin} as:

$$\lambda_{bin} = R\Delta \qquad (4)$$

The number of bins N must be greater than a threshold to allow BSFQ to use all available buffers. If B is the size of the output buffer, then $N\lambda_{bin} \geq B$, or $N \geq \frac{B}{R\Delta}$. If $N < \frac{B}{R\Delta}$, then a portion of the output buffer will not be used by BSFQ. N can be greater than $\frac{B}{R\Delta}$ since the bins are logical. The value of N will affect the ability of BSFQ to detect packets of non-compliant flows: a larger N will allow more packets from non-compliant flows to enter the switch. The buffer management capabilities of BSFQ are similar to those of our pipeline section method in [14] and it can provide lossless guarantee to leaky bucket compliant flows. However, the foci of this paper are on BSFQ's delay guarantee and fairness properties, and the analysis of the buffer management capabilities of BSFQ is outside the scope of the paper.

Before we state and prove Theorem 1, we will present the following corollary that is used in the proof.

Corollary 1: The number of bits of data D admitted into consecutive bins labeled from $[\tau_1, \tau_1 + \Delta)$ to $[\tau_2, \tau_2 + \Delta)$, $\tau_2 > \tau_1$, is bounded by:

$$D \leq (\tau_2 - \tau_1 + \Delta)R + \sum_{all\ f} \ell_f^{max}$$

Proof: Since the virtual time intervals of consecutive bins increases by Δ, we can write $\tau_2 = \tau_1 + (k-1)\Delta$, $k \geq 1$, where k is the number of consecutive bins. By Lemma 1, we have that:

$$\begin{aligned} D &\leq k\Delta R + \sum_{all\ f} \ell_f^{max} \\ &= (\tau_2 - \tau_1 + \Delta)R + \sum_{all\ f} \ell_f^{max} \end{aligned}$$

■

We can now present Theorem 1 which shows that BSFQ belongs to the GR class of schedulers.

Theorem 1: The departure time $L_{BSFQ}(p_f^j)$ of p_f^j in BSFQ is bounded by:

$$L_{BSFQ}(p_f^j) \leq GRC(p_f^j) + \Delta + \frac{\sum_{all\ f} \ell_f^{max}}{R}$$

Proof: The proof is similar to the one presented in [12] for SCFQ. We define $B_f^j = \{n \mid 0 < n \leq j \wedge \tau(A(p_f^n)) \geq vts(p_f^{n-1})\}$. Let k be the largest integer in B_f^j. Since $\tau(A(p_f^1)) \geq vts(p_f^0)$, by the definition of $vts(p_f^0)$, B_f^j contains at least one element and k is always defined. It follows from the definition of B_f^j that:

$$\begin{aligned} \tau(A(p_f^k)) &\geq vts(p_f^{k-1}) \\ \text{and, } \tau(A(p_f^i)) &< vts(p_f^{i-1}), \quad \text{for } k < i \leq j \end{aligned}$$

Therefore, from (1), we have that:

$$vts(p_f^k) = \tau(A(p_f^k)) + \frac{\ell_f^k}{r_f}$$

$$\text{and, } vts(p_f^i) = vts(p_f^{i-1}) + \frac{\ell_f^i}{r_f},\ i = k+1, \ldots, j$$

Thus:

$$vts(p_f^j) - \tau(A(p_f^k)) = \sum_{i=k}^{j} \frac{\ell_f^i}{r_f} \tag{5}$$

Now, assume that p_f^j is entered into the bin with the label $[\tau^*, \tau^* + \Delta)$. At time $A(p_f^k)$ when packet p_f^k arrives, the bin that is being serviced by BSFQ has the label $[\tau(A(p_f^k)), \tau(A(p_f^k)) + \Delta)$. Thus, the maximum amount of data that will be transmitted between p_f^k's arrival and p_f^j's departure is all the data in the bins $[\tau(A(p_f^k)), \tau(A(p_f^k)) + \Delta)$ to $[\tau^*, \tau^* + \Delta)$. According to Corollary 1, the amount of data D stored in these bins is at most:

$$\begin{aligned} D &\leq (\tau^* - \tau(A(p_f^k)) + \Delta)R + \sum_{all\ f} \ell_f^{max} \\ &\leq (vts(p_f^j) - \tau(A(p_f^k)) + \Delta)R + \sum_{all\ f} \ell_f^{max} \end{aligned}$$

From (5), we have that:

$$D \leq \left(\sum_{i=k}^{j} \frac{\ell_f^i}{r_f} + \Delta \right) R + \sum_{all\ f} \ell_f^{max}$$

The data transmission rate is R and therefore, the departure time $L_{BSFQ}(p_f^j)$ of packet p_f^j using a BSFQ scheduler is:

$$\begin{aligned} L_{BSFQ}(p_f^j) &= A(p_f^k) + \frac{D}{R} \\ &\leq A(p_f^k) + \sum_{i=k}^{j} \frac{\ell_f^i}{r_f} + \Delta + \frac{\sum_{all\ f} \ell_f^{max}}{R} \end{aligned} \tag{6}$$

From the definition of $GRC(p_f^j)$ in (3), we have:

$$GRC(p_f^k) \geq A(p_f^k) + \frac{\ell_f^k}{r_f}$$

$$\text{and, } GRC(p_f^i) \geq GRC(p_f^{i-1}) + \frac{\ell_f^i}{r_f}, \text{ for } k < i \leq j$$

Thus:

$$GRC(p_f^j) \geq A(p_f^k) + \sum_{i=k}^{j} \frac{\ell_f^i}{r_f} \tag{7}$$

From (6) and (7), it follows that:

$$L_{BSFQ}(p_f^j) \leq GRC(p_f^j) + \Delta + \frac{\sum_{all\ f} \ell_f^{max}}{R}$$

■

B. Fairness Guarantee

We show that the difference between the normalized service of any two flows in BSFQ is bounded. Let $L(p_f^j)$ denote the departure time of packet p_f^j of flow f. At time $t = L(p_f^j)$,

BSFQ is servicing a bin with label $[\tau(L(p_f^j)), \tau(L(p_f^j)) + \Delta)$. Since p_f^j is contained by this bin, we have that:

$$\tau(L(p_f^j)) \leq vts(p_f^j) < \tau(L(p_f^j)) + \Delta \tag{8}$$

for any packet p_f^j. We need the following auxiliary lemma to show the fairness property of BSFQ.

Lemma 2: If flow f is backlogged at time $t = A(p_f^j)$, then

$$vts(p_f^j) = vts(p_f^{j-1}) + \frac{\ell_f^j}{r_k}$$

Proof: If f is backlogged at time $t = A(p_f^j)$, then $A(p_f^j) \leq L(p_f^{j-1})$. Because $\tau(t)$ is a monotonic (step) function of t, it follows that $\tau(A(p_f^j)) \leq \tau(L(p_f^{j-1}))$. From (8), we have that $\tau(A(p_f^j)) \leq vts(p_f^{j-1})$ and therefore:

$$\begin{aligned} vts(p_f^j) &= \max(\tau(A(p_f^j)), vts(p_f^{j-1})) + \frac{\ell_f^j}{r_f} \\ &= vts(p_f^{j-1}) + \frac{\ell_f^j}{r_f} \end{aligned}$$

■

The following theorem shows that BSFQ is fair.

Theorem 2: If flows f and g are backlogged during the interval $[t_1, t_2]$, then

$$| \, w_f(t_1, t_2) - w_g(t_1, t_2) \, | \leq 2 \left(\frac{\ell_f^{max}}{r_f} + \frac{\ell_g^{max}}{r_g} + \Delta \right)$$

Proof: Let $\{p_f^{k_1}, \ldots, p_f^{k_2}\}$ denote the set of packets from f that depart in $[t_1, t_2]$. Figure 4 shows the timing relationships:

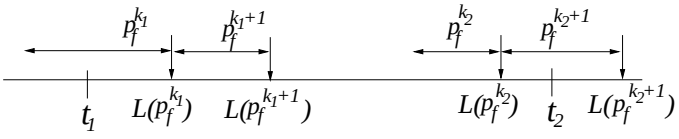

Fig. 4. Packets serviced during the busy period $[t_1, t_2]$

From this figure, we can conclude that the arrival time $A(p_f^i)$ of packet p_f^i must be less than the departure time $L(p_f^{i-1})$ of packet p_f^{i-1}, for $i = k_1 + 1, \ldots, k_2 + 1$, because otherwise flow f would not be backlogged throughout the interval $[t_1, t_2]$. Hence, packets $p_f^{k_1+1}, \ldots, p_f^{k_2+1}$ arrive while f is backlogged. It also follows from Figure 4 that:

$$\sum_{i=k_1+1}^{k_2} \frac{\ell_f^i}{r_f} \leq w_f(t_1, t_2) \leq \sum_{i=k_1}^{k_2+1} \frac{\ell_f^i}{r_f} \tag{9}$$

We first find an upperbound for $w_f(t_1, t_2)$. Since $\tau(\bullet)$ is a non-decreasing function, we have that

$$\tau(t_2) - \tau(t_1) \geq \tau(L(p_f^{k_2})) - \tau(L(p_f^{k_1}))$$

From (8), we have $\tau(L(p_f^{k_2})) > vts(p_f^{k_2}) - \Delta$ and $\tau(L(p_f^{k_1})) \leq vts(p_f^{k_1})$ and therefore:

$$\tau(t_2) - \tau(t_1) \quad > \quad vts(p_f^{k_2}) - vts(p_f^{k_1}) - \Delta$$

Because the packets $p_f^{k_1+1}$, …, $p_f^{k_2}$ arrive while f is backlogged, we have by Lemma 2 that:

$$vts(p_f^{k_2}) = vts(p_f^{k_1}) + \sum_{i=k_1+1}^{k_2} \frac{\ell_f^i}{r_f}$$

and therefore:

$$\tau(t_2) - \tau(t_1) \quad > \quad \sum_{i=k_1+1}^{k_2} \frac{\ell_f^i}{r_f} - \Delta$$

Using (9), we can bound $w_f(t_1, t_2)$ from above by:

$$\begin{aligned} w_f(t_1, t_2) &< \tau(t_2) - \tau(t_1) + \frac{\ell_f^{k_1}}{r_f} + \frac{\ell_f^{k_2+1}}{r_f} + \Delta \\ &\leq \tau(t_2) - \tau(t_1) + 2\frac{\ell_f^{max}}{r_f} + \Delta \end{aligned} \tag{10}$$

To obtain a lower bound for $w_f(t_1, t_2)$, we consider two distinct cases.

Case 1: $vts(p_f^{k_1-1}) \geq \tau(A(p_f^{k_1}))$

Since there is no packet departure in $[t_1, L(p_f^{k_1})]$, we have that $L(p_f^{k_1-1}) \leq t_1$. Therefore:

$$\tau(t_2) - \tau(t_1) \leq \tau(L(p_f^{k_2+1})) - \tau(L(p_f^{k_1-1}))$$

Using (8), we find that $\tau(L(p_f^{k_2+1})) \leq vts(p_f^{k_2+1})$ and $\tau(L(p_f^{k_1-1})) > vts(p_f^{k_1-1}) - \Delta$ and therefore:

$$\tau(t_2) - \tau(t_1) \quad < \quad vts(p_f^{k_2+1}) - vts(p_f^{k_1-1}) + \Delta$$

Since packets $p_f^{k_1}, \ldots, p_f^{k_2+1}$ arrive when f is backlogged, we have that:

$$vts(p_f^{k_2+1}) \quad = \quad vts(p_f^{k_1}) + \sum_{i=k_1+1}^{k_2+1} \frac{\ell_f^i}{r_f}$$

It follows from the case assumption that $vts(p_f^{k_1}) = vts(p_f^{k_1-1}) + \frac{\ell_f^{k_1}}{r_f}$ and therefore:

$$vts(p_f^{k_2+1}) \quad = \quad vts(p_f^{k_1-1}) + \sum_{i=k_1}^{k_2+1} \frac{\ell_f^i}{r_f}$$

and,

$$\tau(t_2) - \tau(t_1) \quad < \quad \sum_{i=k_1}^{k_2+1} \frac{\ell_f^i}{r_f} + \Delta$$

Using (9), we can bound $w_f(t_1, t_2)$ from below by:

$$\begin{aligned} w_f(t_1,t_2) &> \tau(t_2) - \tau(t_1) - \frac{\ell_f^{k_1}}{r_f} - \frac{\ell_f^{k_2+1}}{r_f} - \Delta \\ &\geq \tau(t_2) - \tau(t_1) - 2\frac{\ell_f^{max}}{r_f} - \Delta \end{aligned} \quad (11)$$

Case 2: $vts(p_f^{k_1-1}) < \tau(A(p_f^{k_1}))$

Since f is backlogged at time t_1 and there are no packet departures in $[t_1, L(p_f^{k_1})]$, it follows that $A(p_f^{k_1}) \leq t_1$. Therefore:

$$\begin{aligned} \tau(t_2) - \tau(t_1) &\leq \tau(L(p_f^{k_2+1})) - \tau(A(p_f^{k_1})) \\ &\leq vts(p_f^{k_2+1}) - \tau(A(p_f^{k_1})) \\ &= vts(p_f^{k_1}) + \sum_{i=k_1+1}^{k_2+1} \frac{\ell_f^i}{r_f} - \tau(A(p_f^{k_1})) \end{aligned}$$

It follows from the case assumption that $vts(p_f^{k_1}) = \tau(A(p_f^{k_1})) + \frac{\ell_f^{k_1}}{r_f}$ and therefore:

$$\tau(t_2) - \tau(t_1) \leq \sum_{i=k_1}^{k_2+1} \frac{\ell_f^i}{r_f}$$

and,

$$\sum_{i=k_1+1}^{k_2} \frac{\ell_f^i}{r_f} \geq \tau(t_2) - \tau(t_1) - 2\frac{\ell_f^{max}}{r_f}$$

From (9), we conclude that the lower bound for $w_f(t_1, t_2)$ established in (11) is also valid for the second case. It follows from (10) and (11) that:

$$| \, w_f(t_1,t_2) - w_g(t_1,t_2) \, | \leq 2\left(\frac{\ell_f^{max}}{r_f} + \frac{\ell_g^{max}}{r_g} + \Delta\right)$$

■

IV. Numerical Examples

We have studied the performance of the BSFQ algorithm and compared it with WFQ and DRR. The WFQ algorithm serves as the reference method in the comparison study. The DRR method is selected because it has the same run time complexity as BSFQ for *both* packet processing and connection establishment.

The performance of the BSFQ and DRR methods depend on the setting of some parameters of the methods. In DRR, each flow f is assigned a quantum size q_f which is the amount of credits it receives per round. The reserved bandwidth for flow f is equal to $\frac{q_f}{q_{tot}}R$, where $q_{tot} = \sum_{\text{all flows } j} q_j$. The reserved bandwidth for the flows is unchanged if we use a quantum size $k \times q_j$, $k > 0$, for each flow j. Using smaller quantum sizes will allow DRR to better approximate the WFQ scheduler, but it will also increase the per packet processing cost since the DRR scheduler may need to iterate through a number of rounds without transmitting any packets.

The BSFQ scheduler exhibits a similar behavior as DRR: using a smaller Δ will allow BSFQ to better approximate the WFQ scheduler but it will also increase the likelihood that a bin is empty. The per packet processing cost is increased in this case since the BSFQ scheduler must update the system virtual clock $\tau(t)$. For a fair comparison, we set $\lambda_{bin} = q_{tot}$ so that the amount of data transmitted in each "round" in both schemes are approximately equal.

TABLE I

Flows used in the simulations

	Source rate		Leaky Bucket	
Flow	R_{peak}	R_{avg}	σ	ρ
1,2,3	16Mbps	2Mbps	50KB	2Mbps
4,5,6	25Mbps	5Mbps	100KB	5Mbps
7,8,9	40Mbps	8Mbps	200KB	8Mbps

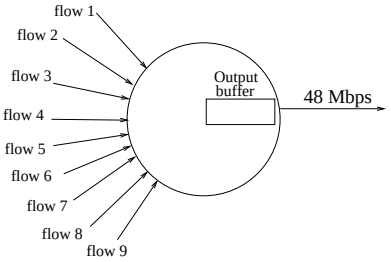

Fig. 5. Network used in the simulation study

We first compare the packet delay with a number of simulation experiments. The network used in the simulation is shown in Figure 5. The transmission rate of the output link is 48 Mbps. There are nine input flows whose properties are shown in Table I. Column 1 in Table I lists the flow indices. Columns 2 and 3 show the peak data rate R_{peak} and the average data rate R_{avg} of each flow. Flows 1, 2 and 3 have low peak and burst rates, flows 4, 5 and 6 have medium peak and burst rates and flows 7, 8 and 9 have high peak and burst rates. Packets have fixed size and each packet is 53 bytes. The packet arrivals of each flow are generated by a Markovian on/off process. When the Markovian process is in the 'on' state, packets are transmitted with a constant rate that is equal to the peak rate R_{peak} in Table I. No packets are transmitted in the 'off' state. The duration of the 'on' and 'off' periods for a flow is such that $R_{avg} = \frac{T_{on} \times R_{peak}}{T_{on}+T_{off}}$, where T_{on} and T_{off} are the average length of the 'on' and 'off' periods, respectively. The average length of an 'on' period in the simulations is set to 1 msec. The Markovian arrival generation process is followed by a leaky bucket rate shaper with the burst size σ and token generation rate ρ given in columns 4 and 5 of Table I. Each type of packet scheduler is presented with the same pattern of packet arrivals and each experiment is run for 5000 (simulation) seconds.

Table II shows the reservations for each flow. For BSFQ,

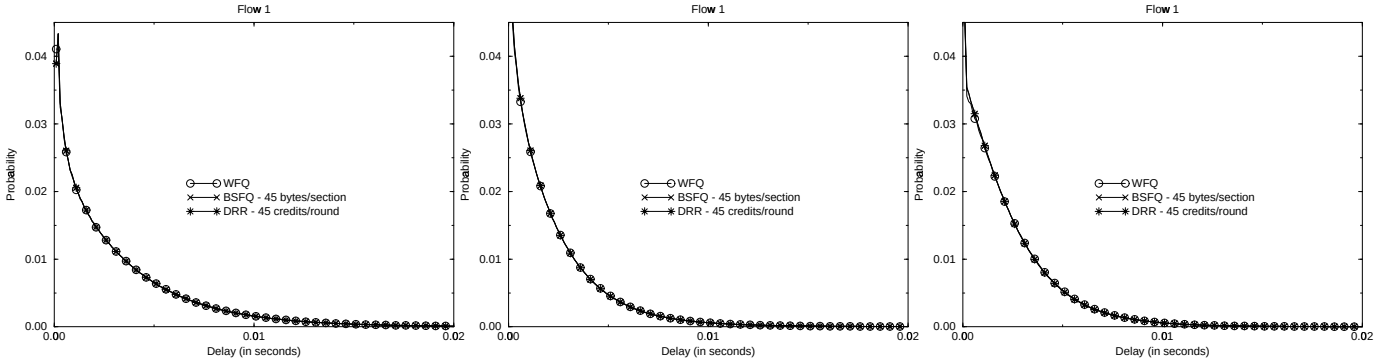

Fig. 6. Delay distribution in Experiment 1 ($Q = 1$ and $\lambda_{bin} = 45$)

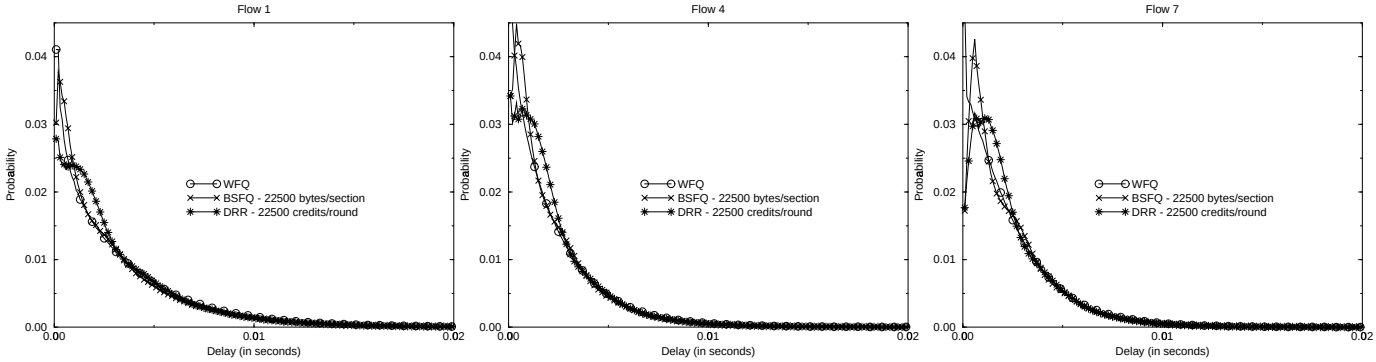

Fig. 7. Delay distribution in Experiment 2 ($Q = 500$ and $\lambda_{bin} = 22500$)

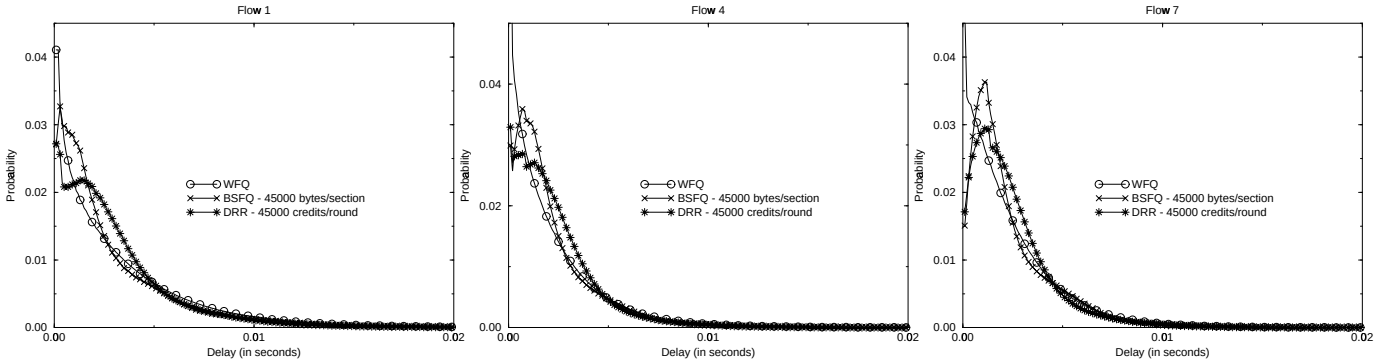

Fig. 8. Delay distribution in Experiment 3 ($Q = 1000$ and $\lambda_{bin} = 45000$)

TABLE II

FLOW RESERVATION PARAMETERS IN EXPERIMENTS

	BSFQ	DRR
Flows	r_f	q_f
1,2,3	2Mbps	$2\times Q$ bytes
4,5,6	5Mbps	$5\times Q$ bytes
7,8,9	8Mbps	$8\times Q$ bytes

the reservation r_f of flow f is equal to its average data rate R_{avg} in Table I. The quantum of flow f in DRR is also proportional to R_{avg}. We vary the parameter Q in DRR and Δ in BSFQ to study their performance. We have performed three sets of experiments using the settings $Q = 1$, $Q = 500$ and $Q = 1000$ in DRR. The corresponding settings for λ_{bin} in BSFQ are $\lambda_{bin} = 45$, $\lambda_{bin} = 22500$ and $\lambda_{bin} = 45000$.

Figure 6 shows the delay distributions in WFQ, DRR and BSFQ for the flows 1, 4 and 7 when $Q = 1$ in DRR and $\lambda_{bin} = 45$ in BSFQ. The graph is a histogram. To generate the histogram, we divided the time axis between $t = 0$ msec and t = 50 msec into intervals of 0.1 msec in length and tallied the number of packets with delay that fall within the interval. Packets with delays larger than 50 msec were tallied in another sum. Each tally is then divided by the total number of packets and the resulting value is an estimate of the probability that the packet delay falls within a given interval. The left most plot in Figure 6 shows the delay distributions of flow 1 using WFQ, DRR and BSFQ. The x-axis in Figure 6 is the packet delay measured in seconds and the y-axis shows the fraction of packets whose delay falls within the given interval. We have shown the delay distributions for delays up to 0.02 second because the tail distributions are virtually identical (including the case where the delay is greater than 0.05 sec). The delay distri-

butions of flows 2 and 3 are similar to that of flow 1 and they are not shown. The center and right plots show the delay distributions for flows 4 and 7, respectively. The distributions of flows 5, 6, 8 and 9 are omitted for the same reason. We can see in Figure 6 that for $Q = 1$ in DRR and $\lambda_{bin} = 45$ bytes in BSFQ, the delay experienced by the packets using either method is virtually identical to that in WFQ. It is evident from Figure 6 that the DRR method using small quantum sizes and the BSFQ method with small λ_{bin} (or Δ) parameters can effectively approximate WFQ.

Figures 7 and 8 show the delay distributions for ($Q = 500$, $\lambda_{bin} = 22500$ bytes) and ($Q = 1000$, $\lambda_{bin} = 45000$ bytes), respectively. We can see that the delay distributions in BSFQ and DRR differ significantly from WFQ for small delays. Also, it appears that the delay distributions in the BSFQ method converges quicker to that of WFQ than DRR. The *total variance distance* [13] is often used to measure how far away two probability distributions are from one another. If p and π are two distributions which put probability mass on a finite set Ω, then the total variance distance between them is equal to:

$$TV(p, \pi) = \frac{1}{2} \sum_{x \in \Omega} \mid p(x) - \pi(x) \mid$$

We have computed the total variance distance of the delay distribution between BSFQ and WFQ, and between DRR and WFQ for each of the above cases. Tables III and IV show the total variance distance between BSFQ and WFQ, and DRR and WFQ, respectively, for the distributions given in Figures 6, 7 and 8. We see that the total variance distance between the delay distribution of BSFQ (DRR) and WFQ for $\lambda_{bin} = 45$ ($Q = 1$) is very small which confirms the fact that the approximation is very good. Tables III and IV show that the total variance distances between BSFQ and WFQ for $\lambda_{bin} = 22500$ and $\lambda_{bin} = 45000$ are smaller than the distances between DRR and WFQ for $Q = 500$ and $Q = 1000$, respectively. Hence, BSFQ provides a better approximation for WFQ in these cases.

TABLE III

TOTAL VARIANCE DISTANCES (BSFQ, WFQ)

Parameter Setting	Flow 1	Flow 4	Flow 7
$\lambda_{bin} = 45$	0.0050	0.0048	0.0069
$\lambda_{bin} = 22500$	0.0430	0.0604	0.0620
$\lambda_{bin} = 45000$	0.0883	0.1028	0.0998

TABLE IV

TOTAL VARIANCE DISTANCES (DRR, WFQ)

Parameter Setting	Flow 1	Flow 4	Flow 7
$Q = 1$	0.0048	0.0049	0.0067
$Q = 500$	0.0738	0.0882	0.0739
$Q = 1000$	0.1180	0.1255	0.1095

An added advantage of BSFQ is the fact that it has a built-in buffer management function. Recall that in BSFQ a packet is discarded when its bin index is equal to or greater than the number of bins N. Packets of a flow that transmits at a rate larger than its reservation will be assigned increasingly larger virtual times and when the virtual time stamp value exceeds $\tau + N\Delta$, they will be discarded. Thus, BSFQ can provide protection against non-conformant flows. However, proving that BSFQ provides a lossless guarantee — similar to the work in [14] — is outside the scope of this paper. We will only demonstrate the benefit of the built-in buffer management function of BSFQ using a simulation experiment.

In the next experiment, we increase the peak and average data rate of flows 7, 8 and 9 in Table I to 50 Mbps and 10 Mbps, respectively. Notice that the total data rate of all flows is equal to 51 Mbps which exceeds the link capacity. The reservation parameters of the flows (see Table II) remains unchanged. The flows 7, 8 and 9 are non-compliant flows in the simulation experiment.

TABLE V

NON-COMPLIANT FLOWS

	Fraction of packets dropped		
	Flow 1	Flow 4	Flow 7
BSFQ	0%	0%	9.89%
DRR	5.43%	4.52%	6.38%

The parameters used in the experiments are $\lambda_{bin} = 45000$ in BSFQ and $Q = 1000$ in DRR. The output buffer size is set to 1 Mbytes and a packet is admitted if there is available space. After a packet has been admitted, it is then processed by the packet scheduler. In DRR, the packet is entered into its queue and will not be discarded. In contrast, the packet is discarded by BSFQ if its bin index exceeds N. Table V shows the percentage of packets dropped by BSFQ and DRR. The table only shows the fraction of the packets dropped from flows 1, 4 and 7. The results for flows 2 and 3, 5 and 6 and 8 and 9, are similar to that of flow 1, 4 and 7, respectively. We can see that BSFQ only drops packets from the non-compliant flows. In contrast, all flows in DRR suffer packet loss.

V. CONCLUSION

We have presented the BSFQ packet scheduling algorithm that combines the strengths of frame-based and priority-based schedulers. BSFQ is a frame-based method that transmits all packets in the current bin in each round. Packets are sorted into the bins based on their assigned virtual time stamps and within one bin, packets are transmitted in the FIFO manner. BSFQ is highly scalable, having constant run time complexity for packet processing, as well as for connection establishment.

The virtual time interval parameter Δ has a significant impact on BSFQ's performance. For very large values of Δ, the performance of BSFQ is identical to FIFO, while for small values of Δ, BSFQ is more akin to SCFQ. There are a number of factors that have to be considered in choosing the value of Δ.

The amount of state information maintained is inversely proportional to Δ and a small Δ will increase the number of bins needed. The efficiency of BSFQ also decreases when Δ is decreased because it increases the likelihood of that some bins are empty. This is similar to DRR when small quantum values are used and no packets are sent in some rounds. Determining an optimal value for Δ is beyond the scope of this paper.

We have shown that BSFQ can provide end-to-end delay guarantee and fairness to flows that share a common link. BSFQ also has a built-in buffer management function that can insulate conformant flows from non-compliant traffic. Results of a simulation study show that BSFQ can provide better approximation for WFQ than DRR using similar operational parameters. In summary, BSFQ has many desirable strengths, including scalability, rate and fairness guarantee and built-in buffer management, to provide quality of service in high-speed networks.

REFERENCES

[1] L. Kleinrock, *Queueing Systems, Volume 1: Theory*, John Wiley & Sons, New York, 1975, ISBN 0-471-49110-1.

[2] A. K. J. Parekh and R. G. Gallager *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case*, IEEE/ACM Transaction on Networking, vol. 1, no. 3, pp. 344–357, 1993.

[3] H. Zhang, *WF2Q: Worst-case Fair Weighted Fair Queueing*, IEEE Infocom 1996, pp. 120–128, 1996.

[4] H. Zhang, *Service Disciplines For Guaranteed Performance Service in Packet-Switching Networks*, Proceedings of the IEEE, 83(10), Oct, 1995.

[5] D. Stiliadis and A. Varma, *Rate-proportional Servers: A Design Methodology for Fair Queueing Algorithms*, IEEE/ACM Transactions on Networking, vol. 6, No. 2, pp. 164–174, 1998.

[6] L. Zhang, *Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks*, ACM SIGCOMM 1990, pp. 19–29, 1990.

[7] S. J. Golestani, *A Self-Clocked Fair Queueing Scheme for Broadband Applications*, IEEE Infocom 1994, pp. 636–646, 1994.

[8] S. Suri, G. Varghese and G. Chandranmenon, *Leap Forward Virtual Clock: A New Fair Queueing Scheme with Guaranteed Delays and Throughput Fairness*, IEEE Infocom 1997, pp. 557–565, 1997.

[9] M. Shreedhar and G. Varghese, *Efficient Fair Queueing using Deficit Round Robin*, IEEE/ACM Transactions on Networking, vol. 4, no. 3, pp. 375–385, 1998.

[10] N. Matsufuru and R. Aibara *Efficient Fair Queueing for ATM Networks using Uniform Round Robin*, IEEE Infocom 1999, New York, March 1999.

[11] G. G. Xie and S. Lam, Delay Guarantee of Virtual Clock Server, IEEE/ACM Transactions on Networking, vol. 3, no. 6, pp. 683–689, 1995.

[12] P. Goyal, S. Lam and H. Vin, Determining End-to-End Delay Bounds In Heterogeneous Networks, Proc. 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video, Durham, New Hampshire, April 18-21, 1995, pp. 287-298.

[13] S. T. Rachev, *Probability Metrics and the Stability of Stochastic Models*, Wiley Series in Probability and Mathematical Statistics, Chichester, New York, Wiley, 1991.

[14] S. Y. Cheung and C. S. Pencea, *Pipelined Sections: A New Buffer Management Discipline for Scalable QoS Provision*, IEEE Infocom 2001.