This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2011 proceedings

Worst-Case Fair Bin Sort Queuing (WBSQ): An $O(1)$ Worst-Case Fair Scheduler

Zyad A. Dwekat
Sprint Network Services, Raleigh, NC
George N. Rouskas
Department of Computer Science, North Carolina State University, Raleigh, NC, 27695

***Abstract*—The design of packet schedulers involves a tradeoff between implementation complexity, on one hand, and delay and fairness guarantees, on the other. In this paper, we present worst-case fair bin sort queuing (WBSQ), a new scheduler that has good worst-case fairness and delay properties, yet has low complexity and is amenable to simple hardware implementation. WBSQ achieves this performance by combining features of BSFQ and WF^{2}Q+. We establish the worst-case fairness and delay properties of WBSQ through both analysis and simulation.**

I. Introduction

The Internet has evolved into a ubiquitous global communication medium that carries a constantly evolving traffic mix that is becoming richer as innovation and technology improvements spawn new telecommunications applications and services. Hence, the network must support some form of quality of service (QoS) functionality in order to serve large numbers of heterogeneous users and applications with a wide range of requirements in terms of throughput and delay. In packet-switched networks, the link scheduling discipline is a central component of the QoS architecture.

Packet schedulers can be classified as either *timestamp-based* or *frame-based*. Timestamp schedulers emulate the ideal but unimplementable generalized processor sharing (GPS) algorithm by maintaining a virtual time function. For example, weighted fair queuing (WFQ) [6] assigns a timestamp to each packet based on the virtual time value at the time of arrival, and transmits packets in increasing order of timestamp. Frame-based schedulers operate by dividing time into frames. Within each frame, flows are mapped to time slots of fixed or variable length and are served in a round-robin manner.

In general, timestamp schedulers have good delay and fairness properties but high complexity, hence they have found limited deployment in high-speed routers. On the other hand, frame-based schedulers such as deficit round robin (DRR) [8] are easy to implement and have been widely deployed in commercial routers. However, frame-based schedulers have poor behavior in terms of delay bound and output burstiness.

More recently, *hybrid* designs such as smoothed round robin (SRR) [3] and stratified round robin (S-RR) [7] incorporate some elements of timestamp schedulers into a frame-based scheme. In doing so, hybrid schedulers attempt to combine the best of both worlds, i.e., improve the delay and output burstiness of frame-based scheduler while maintaining low implementation complexity. The worst-case fair bin sort queuing (WBSQ) scheduler we present in this work is a hybrid scheduler that, unlike similar schedulers, is worst-case fair.

This paper is organized as follows. In Section II, we review the complexity of timestamp schedulers, and in Section III, we describe the operation of the WBSQ scheduler. In Section IV, we establish theoretically the properties of the scheduler, and we present experimental results in Section V. We conclude the paper in Section VI.

II. Complexity of Timestamp Schedulers

Timestamp schedulers maintain a virtual time function $V(t)$ to track the progress of GPS. Suppose that the k-th packet of a flow i arrives at time a_i^k, the packet length is L_i^k and the rate of the flow is r_i. Let S_i^k and F_i^k denote the virtual times at which this packet begins and completes service, respectively, and let $F_i^0 = 0$ for all flows i. At the time of arrival $t = a_i^k$, WFQ and its variants compute these values as [6]:

$$S_i^k = \max\{F_i^{k-1}, V(a_i^k)\} \quad (1)$$

$$F_i^k = S_i^k + \frac{L_i^k}{r_i} \quad (2)$$

The complexity of such schedulers arises from two factors.

1) *Packet sorting.* The scheduler selects among the head-of-line packets of the backlogged flows the one with the smallest finish time (2) to serve next; for n flows, this operation takes time $O(\log n)$ using a priority queue.
2) *Virtual time computation.* The virtual time is updated periodically by evaluating the corresponding function; the worst-case complexity of this operation is $O(n)$ [5].

One of the differentiating characteristics of timestamp scheduler variants is their approach to alleviating the complexity of these two operations.

The WF^{2}Q+ scheme [1] improves upon WFQ in two ways. First, it uses a different, approximate, virtual time function, shown in (3), that can be evaluated in time $O(\log n)$:

$$V(t+\tau) = \max\{V(t) + \tau, \min_{i=1,\ldots,n} \{S_i\}\}, \quad (3)$$

where S_i is the start time of the head-of-line packet of flow i. In other words, the scheduler maintains only one pair of start and finish times (S_i, F_i) for each flow i, those corresponding to the head-of-line packet. Second, when the server chooses the next packet to transmit at time t, rather that selecting among all packets that are backlogged at time t (as WFQ

978-1-61284-233-2/11/$26.00 ©2011 IEEE

does), it selects only from *eligible* packets, i.e., packets that would have started receiving service in the corresponding GPS system at t. WF^{2}Q+ achieves tight delay bounds and good worst-case fairness properties with an overall algorithmic complexity of $O(\log n)$.

Bin sort fair queuing (BSFQ) [2] takes a different approach to reducing the complexity. Specifically, virtual time is divided into bins (slots) of length δ, and the scheduler maintains a virtual system clock that is equal to the left endpoint of the current bin. Arriving packets are assigned a timestamp that is equal to the finish time in expression (2). Packets with timestamps that fall within the same bin, are inserted in a FIFO queue associated with this bin. In other words, there is no sorting of packets that have finish times "close" to each other, i.e., within the length δ of a bin, hence this "bin sorting" operation takes $O(1)$ time. The BSFQ scheduler does not need to evaluate a virtual time function similar to (3); instead, once the FIFO queue of the current bin is served, virtual time advances by δ and the scheduler serves the FIFO queue of the next bin. Consequently, the operations of BSFQ are scalable and easy to implement in hardware. Its fairness and delay guarantees depend strongly on the value of δ: smaller δ values result in better fairness and delay guarantees, but the amount of state information that the scheduler needs to maintain, as well as its complexity, increases. Furthermore, the worst case delay of any packet is proportional to the number n of flows.

III. Worst-Case Fair Bin Sort Queuing (WBSQ)

The worst-case fair bin sort queuing (WBSQ) scheduler combines elements of the BSFQ and WF^{2}Q+ schemes to achieve worst-case fairness at $O(1)$ complexity. Specifically, WBSQ achieves the worst-case fairness and delay bound properties of WF^{2}Q+ at the same complexity as BSFQ by incorporating the following key features:

1) *Quantization of virtual time.* In WBSQ, virtual time $V(t)$ is quantized such that $V(t) = m\delta, m = 0, 1, 2, \ldots$, where δ is the bin size as in BSFQ. Hence, WBSQ completely avoids the overhead of computing a complex virtual time function at packet arrival and departure instants.
2) *Bin sorting of virtual finish times.* WBSQ avoids the $O(\log n)$ complexity of selecting the next packet to serve by using a bin sorting operation to order packets according to their finish times. This mechanism is similar to the one introduced by BSFQ.
3) *Observing packet eligibility.* As in WF^{2}Q+, the WBSQ scheduler maintains start time and finish time timestamps only for the head-of-line packet of each flow. Furthermore, the scheduler only serves *eligible* packets, i.e., those that would have started service under GPS, resulting in much better fairness than BSFQ that does not have a mechanism for checking eligibility.

A. Data Structures for Bin Sorting

The WBSQ scheduler employs the concept of bin sorting introduced by BSFQ [2] to eliminate the need for sorting virtual finish times and computing the virtual time function. As shown in Figure 1, WBSQ uses one circular buffer that is divided into M virtual time bins, where M is a system design parameter. Each bin represents an interval of virtual time of length δ; the bin width δ is an important design parameter whose impact will be investigated in the following section. Let $V(t) = V$ be the current virtual time. Then, the M bins correspond to virtual time intervals $[V + m\delta, V + (m+1)\delta), m = 0, 1, \cdots, M-1$. In other words, the virtual time intervals represented by the bins are non-overlapping and their union spans a continuous range of the virtual time space of length $M \times \delta$ starting at the current time V. Hence, we will use the convention that the bin with index $m = 0$ always represents the current virtual time.

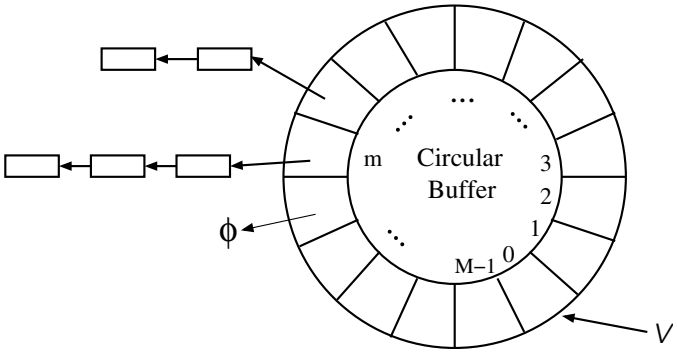

Fig. 1. Data structures of the WBSQ scheduler

Each bin of the circular buffer has one FIFO queue that is implemented as a linked list of records. Consider the m-th bin representing the virtual time interval $[V + m\delta, V + (m + 1)\delta), m = 0, 1, \cdots, M-1$. Each record of the FIFO queue of this bin corresponds to a flow that has a head-of-line packet with finish time that falls in the interval $[V+m\delta, V+(m+1)\delta)$. Note that, by the operation of the scheduler described below, each flow will have a record present in the queue of at most one bin in any given time. This is a major difference to BSFQ in which a flow may have multiple packets queued in the circular buffer.

B. Scheduler Operation

The operation of the scheduler is fully defined by the actions taken whenever a relevant event takes place. The relevant events occur when (1) a packet arrives, (2) a packet reaches the head of its flow's queue, and (3) a packet departs (i.e., completes service), and are examined in this order next.

1) Packet Arrival: When packet k of flow i arrives at time t, it is inserted at the tail of this flow's queue and its start S_i^k and finish F_i^k times are computed according to expressions (1) and (2), respectively. If flow i was active just prior to the arrival, then no other action is taken, and the virtual start and finish times of this packet are not considered by the scheduler. If, on the other hand, flow i was inactive prior to the arrival, then the arriving packet reaches the head of this flow's queue at time t and additional actions are taken as described next.

2) Packet Reaches The Head Of Its Queue: Suppose that at time t, packet k of flow i reaches the head of flow i's queue.

The start time S_i and finish time F_i of flow i are updated to the start S_i^k and finish F_i^k times of this packet, respectively. Now note that, at time t, no record of flow i exists in any FIFO queue of any bin. If flow i was inactive just prior to t, no record would exist. If, on the other hand, flow i was active, the fact that packet k reached the head of the queue at time t implies that the previous head-of-line packet of flow i was the one that completed service just before time t. In this case, the previous record of flow i was removed at the time the previous head-of-line packet departed, as we explain shortly.

At time t, a new record of flow i is created and inserted at the queue of one bin. Let m be an integer such that $V + m\delta \leq F_i^k < V + (m+1)\delta$. The record consists of a pointer to the head-of-line packet of flow i and is inserted at the tail of the FIFO queue of bin m.

3) Packet departure: When a packet of some flow i completes service, two main actions take place: the record of the flow is removed from the current bin (i.e., the one with index $m = 0$, representing the current virtual time), and a new packet is selected for service. Removing the record (i, S_i) is easy since, as we shall see, it is the first record of the FIFO queue in the current bin.

Next, the scheduler must select a new packet to serve. To do so, it inspects the FIFO queue (i.e., after the removal of the previous record) of the current bin. If the FIFO queue is not empty, then the scheduler starts serving the packet indicated by the record at the head of the FIFO queue. If, however, the queue is empty, then virtual time advances by δ and bin indexes shift by one such that the next bin (previously bin $m = 1$) becomes the current one (i.e., the new bin indexed $m = 0$). As a result, the bin that was previously the current one becomes the last one in the circular buffer, such that the buffer again represents a range of virtual times of length $M \times \delta$ starting at the current time. The scheduler then proceeds to serve the packet indicated by the record at the head of the new current bin's queue; if the bin is empty, the process of advancing the virtual time by an amount of δ repeats until a bin with a non-empty queue is found.

IV. Fairness and Delay Properties

We first observe that the WBSQ scheduler always serves eligible packets *within the granularity of the virtual time quantum* δ. Let V be the current virtual time, which, by definition, is the left endpoint of the virtual time interval $[V, V + \delta)$ represented by the current bin with index $m = 0$. Therefore, any packet served from this bin is such that $F_i \in [V, V + \delta)$, where i is the flow to which the packet belongs. Since $S_i < F_i$, there are two possibilities: either $S_i \leq V$, in which case the flow (and packet) is clearly eligible for service; or $V < S_i < F_i < V + \delta$, in which case, since the WBSQ scheduler does not distinguish between virtual times in the interval $[V, V+\delta)$, the packet is considered eligible. Consequently, the bin width δ represents the error in determining packet eligibility within the WBSQ system.

Let L_{max} denote the maximum packet length of any flow, r denote the link rate, and r_{min} be the minimum rate allocated to any flow in the WBSQ system; r_{min} represents the granularity of rate allocation, and is a constant for a given scheduler. We assume that the link is not oversubscribed, i.e., $\sum_i r_i \leq r$. Let C be a constant such that:

$$\frac{L_{max}}{r_{min}} = C\delta \tag{4}$$

We have the following result.

Lemma 4.1: The maximum amount of work W present in the circular buffer at any point in time is bounded by:

$$W \leq rC\delta \tag{5}$$

Proof. There is at most one packet from each flow i represented in the circular buffer. Let k_i be the index of the packet of flow i in the buffer. Using (4), we obtain:

$$\begin{aligned} W &= \sum_i L_i^{k_i} \leq \sum_i L_{max} = C\delta \sum_i r_{min} \\ &\leq C\delta \sum_i r_i = rC\delta. \end{aligned} \tag{6}$$

■

A. Delay Bound

We have the following result regarding the relative departure times of packets in the WF^{2}Q+ and WBSQ systems.

Corollary 4.1: Consider a flow i that is eligible for service. Let $d_i^{WF^2Q+}$ denote the departure time of the head-of-line packet of flow i under the WF^{2}Q+ scheduler, and d_i^{WBSQ} the departure time of the same packet in the corresponding WBSQ system. Then:

$$d_i^{WBSQ} - d_i^{WF^2Q+} \leq C\delta \tag{7}$$

where δ is the bin width.

Proof. Recall that WBSQ calculates packet finish times using the same expression (2) as WF^{2}Q+. Consequently, any difference in the departure times of a given packet under the two systems is due to differences in the scheduling of the corresponding flow. WF^{2}Q+ schedules eligible flows in strictly increasing order of their finish times. The WBSQ scheduler, on the other hand, serves flows within the same bin in a FIFO order. As a result, a flow with smaller finish time may receive service before one with a higher finish time. However, such departure from the strictly increasing order of finish times is limited to flows that fall within the *same* bin. In other words, WBSQ will not serve a flow from a higher bin until all flows in the current bin are served.

In the worst case scenario, a flow i in the current bin would be the first one to be served under WF^{2}Q+ but the last flow of that bin to be served by WBSQ, *and* all eligible flows fall within the same (current) bin. Expression (5), provides an upper bound on the amount of service that flows within the current bin may receive before flow i is served under WBSQ. Therefore, to obtain a bound on the difference between the departure times of flow i under WBSQ and WF^{2}Q+ we may simply divide both sides of (5) by the link rate r, which yields the desired result in (7). ■

B. Proportional Fairness

The fairness measure of Golestani [5] essentially requires that the difference between the normalized service received by any two backlogged flows i and j, over any time period (t_1, t_2), be bounded by a small constant. We have the following result for WBSQ.

Theorem 4.1 (Golestani fairness): In any time period (t_1, t_2) during which flows i and j are backlogged,

$$\left| \frac{S_i(t_1,t_2)}{r_i} - \frac{S_j(t_1,t_2)}{r_j} \right| \leq 3\left(\frac{L_{max}}{r_i} + \frac{L_{max}}{r_j} + \delta \right) \quad (8)$$

where $S_i(t_1, t_2)$ is the amount of data sent by flow i during the time period (t_1, t_2).

Proof. Due to page constraints, the proof is omitted, but can be found in [4]. ■

C. Worst-Case Fairness

The worst-case fairness measure of Bennett-Zhang is a more refined notion of fairness. Rather than comparing the relative amounts of service received by two flows i and j, it compares the service received by a single flow i to the service it would receive in the ideal case, i.e., when i has exclusive access to an output link of rate r_i. Suppose that a packet belonging to flow i arrives, creating a total backlog of q_i bits in i's queue. In the ideal case, it would take an amount of time equal to q_i/r_i to drain the backlog. Define D_i as the additional amount of time (i.e., in excess of q_i/r_i) that it would take to clear the backlog under some scheduler. The fairness measure of Bennett-Zhang requires a bound on the value of D_i. Theorem 4.2 gives the bound on this value for WBSQ.

Theorem 4.2 (Bennett-Zhang fairness): The value of D_i^{WBSQ} is bounded by:

$$D_i^{WBSQ} \leq C\delta + \frac{L_{max}}{r} \quad (9)$$

Proof. The proof follows directly from Corollary 4.1 and the fact that [1]:

$$D_i^{WF^2Q+} \leq \frac{L_{max}}{r}. \quad (10)$$

From this expression and (7), we obtain:

$$D_i^{WBSQ} \leq C\delta + D_i^{WF^2Q+} = C\delta + \frac{L_{max}}{r} \quad (11)$$

■

V. Experimental Results

In this section we report the results of simulation experiments designed to (1) investigate the fairness and delay properties of WBSQ in practical situations, and (2) compare WBSQ to the WF^{2}Q+ and BSFQ disciplines. All experiments were performed using ns-2, to which we added new WBSQ and BSFQ queuing classes. While we carried out extensive simulations, due to page constraints we will only report the results of two representative experiments, one for short-term throughput and one for end-to-end average delay variation. Figure 2 shows the network topology we used in the experiments. All the links have a bandwidth of 2 Mbps and a propagation delay of 1 ms. In all experiments, there are 20 CBR flows from node S to node D. The average flow rates range from 10-280 Kbps. The 20 flows start at random times between $t = 0$ and $t = 2$ sec, and stop at $t = 10$ sec.

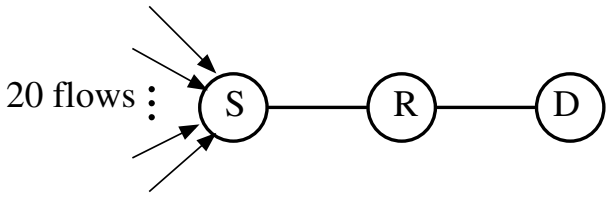

Fig. 2. Network topology used for the simulation

In the first set of experiments we calculated the average flow throughput over 100 ms intervals, and we varied the bin width $\delta = 1000, 600, 200$. Figures 3-5 plot the throughput of a representative flow against time, and each corresponds to a different value of δ. Each figure consists of three curves corresponding to three schedulers, WF^{2}Q+, BSFQ, and WBSQ; note that only BSFQ and WBSQ are affected by the value of the bin width.

As we can see from the three figures, WF^{2}Q+ has the most consistent throughput behavior, staying close to the 120 Kbps rate at which this flow transmits. At the other extreme, BSFQ has the worst throughput performance with a high variation in throughput over time. Although the WBSQ scheduler does exhibit throughput variation, it better tracks WF^{2}Q+ and the throughput is close to the the rate of the flow.

We also observe that, as the bin width decreases (from 1000 in Figure 3 to 600 in Figure 4 and 200 in Figure 5), the variation in throughput decreases for both BSFQ and WBSQ. Nevertheless, the throughput variation for BSFQ remains high even for $\delta = 200$, whereas the behavior of WBSQ quickly approaches that of WF^{2}Q+. We conclude that WBSQ performs significantly better than BSFQ, and, despite its low complexity, it closely tracks the performance of WF^{2}Q+.

In the second experiment, we measured the average variation in end-to-end delay over all packets of the same flow as in the first experiment. Figure 6 plots the delay variation against the value of the bin width δ for the same three schedulers. As we can see, WF^{2}Q+ has the lowest delay variation and, as expected, its behavior is independent of the bin width. BSFQ has the highest average delay variation across all bin width values. On the other hand, WBSQ has much better average delay variation than BSFQ, and its performance improves as the bin width decreases.

Based on the above results, we conclude that WBSQ represents a good tradeoff between complexity and performance, approaching the fairness and delay behavior of WF^{2}Q+ at the low implementation complexity of BSFQ.

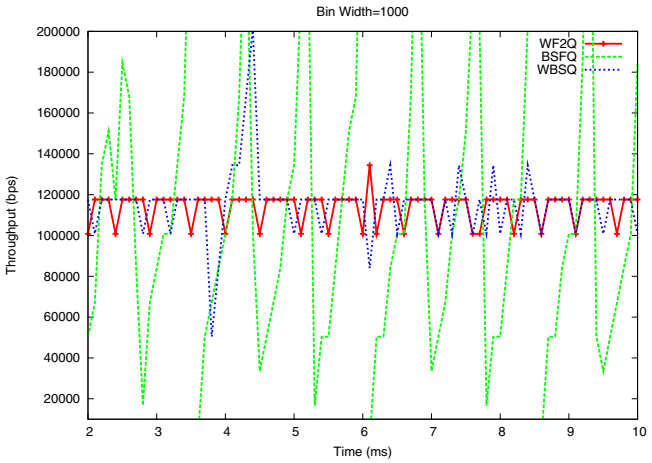

Fig. 3. Short-term throughput for δ=1000

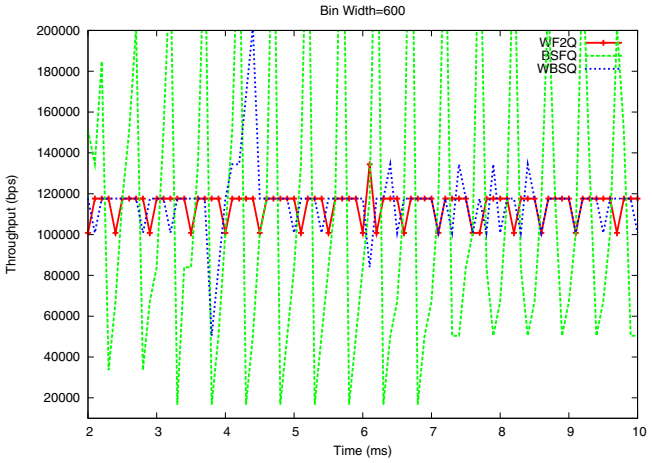

Fig. 4. Short-term throughput for δ=600

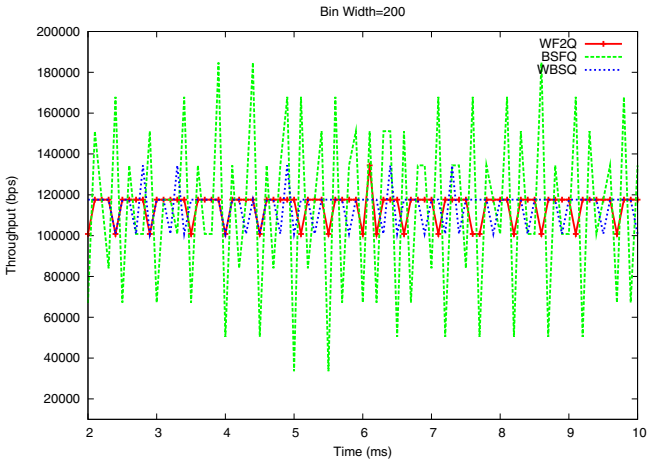

Fig. 5. Short-term throughput for δ=200

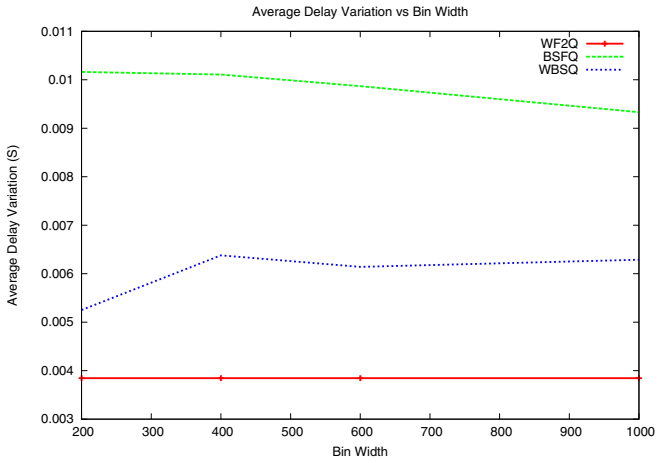

Fig. 6. Average delay variation as a function of bin width δ

VI. Concluding Remarks

We have presented WBSQ, a new packet scheduler that combines the bin sort concept of BSFQ with the simplified virtual time calculation of WF^{2}Q+. We have established theoretically that WBSQ has constant proportional and worst-case fairness indexes. Simulation results demonstrate that the short-term throughput and average delay variation behavior of WBSQ is close to that of WF^{2}Q+, especially for smaller bin width values. WBSQ has low complexity and its operations may be easily implemented in hardware. Therefore, we believe that employing WBSQ scheduling within high-speed routers will enable network operators to enhance their ability to offer and guarantee a wide range of services.

References

[1] J. C. R. Bennett and H. Zhang. Hierarchical packet fair queuing algorithms. In *Proceedings of ACM SIGCOMM '96*, pages 143–156, August 1996.

[2] S. Cheung and C. Pencea. BSFQ: bin sort fair queuing. In *Proceedings of IEEE INFOCOM '02*, 2002.

[3] G. Chuanxiong. SRR, an $O(1)$ time complexity packet scheduler for flows in multi-service packet networks. In *Proceedings of ACM SIGCOMM '01*, pages 211–222, August 2001.

[4] Zyad Dwekat. *Practical Fair Queueing Schedulers: Simplification Through Quantization*. PhD thesis, North Carolina State University, Raleigh, NC, August 2009.

[5] S. Golestani. A self-clocked fair queuing scheme for broadband applications. In *Proceedings of IEEE INFOCOM '94*, pages 636–646, 1994.

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

[7] S. Ramabhadran and J. Pasquale. Stratified round robin: A low complexity packet scheduler with bandwidth fairness and bounded delay. In *Proceedings of ACM SIGCOMM '03*, pages 239–249, August 2003.

[8] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round robin. In *Proceedings of ACM SIGCOMM '95*, 1995.