A New Starting Potential Fair Queuing Algorithm with O(1) Virtual Time Computation Complexity

Dong-Yong Kwak, Nam-Seok Ko, Bongtae Kim, and Hong-Shik Park

In this paper, we propose an efficient and simple fair queuing algorithm, called new starting potential fair queuing (NSPFQ), which has O(1) complexity for virtual time computation and also has good delay and fairness properties. NSPFQ introduces a simpler virtual time recalibration method as it follows a rate-proportional property. The NSPFQ algorithm recalibrates the system virtual time to the minimum virtual start time among all possible virtual start times for head-of-line packets in backlogged sessions. Through analysis and simulation, we show that the proposed algorithm has good delay and fairness properties. We also propose a hardware implementation framework for the scheduling algorithm.

Keywords: Fair queuing, packet scheduling algorithm, WFQ.

I. Introduction

The packet scheduling algorithm is very important within individual switches or routers in high-speed integrated services packet networks for providing a wide range of quality-of-service guarantees. The function of a scheduling algorithm is to select, for each outgoing link of the switch, the packet to be transmitted in the next cycle from the available packets belonging to the flows sharing the same output link.

Until now, the literature has presented many packet scheduling algorithms, which are also called fair queuing algorithms, [1]-[16]. Among them, weighted fair queuing (WFQ) [2] is an ideal packet scheduling algorithm for its delay and fairness properties. However, the timestamp computation in the WFQ scheduler serving N sessions has a complexity of O(N) per packet transmission time and this makes its implementation difficult. Recently, many algorithms have been proposed to simplify the implementation of WFQ, such as self-clocked fair queuing (SCFQ) [10], frame-based fair queuing (FFQ) [11], starting potential fair queuing (SPFQ) [11], and minimum delay self-clocked fair queuing (MD-SCFQ) [13]. However, all those algorithms have their own shortcomings, which are explained in section II.

In this paper, we propose an efficient and simple fair queuing algorithm, called new starting potential fair queuing (NSPFQ), which has O(1) complexity for virtual time computation and good delay and fairness properties that are comparable to those of WFQ.

This paper is organized as follows. In section II, we review the existing fair queuing algorithms and their problems. Section III presents the concept and operational principle and the performance analysis of the proposed algorithm. We also provide an extended algorithm to get better fairness on a

Manuscript received Aug. 2, 2002; revised Apr. 14, 2003.

Dong-Yong Kwak (phone: +82 42 860 5148, email: dykwak@etri.re.kr), Nam-Seok Ko (email: nsko@etri.re.kr), and Bongtae Kim (email: bkim@etri.re.kr) are with Network Technology Laboratory, ETRI, Daejeon, Korea.

Hong-Shik Park (email: hspark@icu.ac.kr) is with Information and Communications University, Daejeon, Korea.

statistical base making the fairness index unchanged. In section IV, we propose the hardware implementation framework of the proposed scheduler. Section V gives the simulation results. Finally, section VI presents our conclusions.

II. Fair Queuing Algorithms

1. Background

In general, schedulers can be characterized as work-conserving or nonwork-conserving. A scheduler is work-conserving if the server is never idle when a packet is buffered in the system. A nonwork-conserving server may remain idle even if there are available packets to transmit. A server may, for example, postpone the transmission of a packet when it expects a higher priority packet to arrive soon, even though it is currently idle. Nonwork-conserving algorithms are used to control delay jitter by delaying packets that arrive early. Work-conserving servers always have lower average delays than nonwork-conserving servers. Examples of work-conserving schedulers include generalized processor sharing (GPS) [9], WFQ, virtual clock [2], weighted round robin (WRR) [3], and deficit round robin (DRR) [4]. On the other hand, hierarchical round robin (HRR) [6], stop-and-go queuing [7], and jitter earliest due date (Jitter-EDD) [5] are nonwork-conserving schedulers.

Another classification of schedulers is based on their internal structure [8]. According to this classification, there are two main architectures: sorted-priority and frame-based. In a sorted-priority scheduler, there is a global variable, usually referred to as the virtual time (also known as system potential), associated with each outgoing link of the switch. Each time a packet arrives or gets serviced, this variable is updated. A timestamp, computed as a function of this variable, is associated with each packet in the system. Packets are sorted based on their timestamps and are transmitted in that order. WFQ is the representative algorithm for this architecture. Two factors determine the implementation complexity of all sorted-priority algorithms. First, the complexity of updating the priority list and selecting the packet with the highest priority is at least O(log N), where N is the number of connections sharing the outgoing link. The second is the complexity of calculating the timestamp associated with each packet; this factor depends heavily on the algorithm. In a frame-based scheduler, time is split into frames of fixed or variable length. Reservations of sessions are made in terms of the maximum amount of traffic the session is allowed to transmit during a frame period. Hierarchical round robin and stop-and-go queuing are frame-based schedulers that use a constant frame size. As a result, the server may remain idle if sessions transmit less traffic than their reservations over the duration of a frame. In contrast, WRR and DRR schedulers allow the frame size to vary within a maximum. Thus, if the traffic from a session is less than its reservation, a new frame can be started early. Therefore, both of these schedulers are work-conserving.

In this paper, we focus on the sorted-priority and work conserving algorithm. GPS is an ideal scheduling discipline among such algorithms. GPS multiplexing is defined with respect to a fluid model, where packets are considered to be indefinitely divisible. The share of bandwidth reserved by session i is represented by a real number. Let $B(\tau,t)$ be the set of connections that are backlogged in the interval $(\tau,t]$. If r is the rate of the server, the service offered to a connection i that belongs to $B(\tau,t)$, $W_i(\tau,t)$, is proportional to r_i as follows:

$$W_i(\tau,t) \geq \frac{r_i}{\sum_{j\in B(\tau,t)} r_j} r(t-\tau).$$

The minimum service that a connection can receive in any interval of time is

$$\frac{r_i}{\sum_{j=1}^{N} r_j} r(t-\tau),$$

where N is the maximum number of connections that can be backlogged in the server at the same time. Thus, GPS serves each backlogged session with a minimum rate equal to its reserved rate at each instant; in addition, the excess bandwidth available from sessions not using their reservations is distributed among all the backlogged connections at each instant in proportion to their individual reservations. This results in perfect isolation, ideal fairness, and low end-to-end session delays.

The WFQ, or packet-by-packet GPS (PGPS), algorithm [9] is the packet-by-packet equivalent of GPS, that is, it derives the system virtual time from the background simulation of a GPS server. The system virtual time $v(t)$ of WFQ evolves as that of the corresponding GPS system, whose derivative is as follows,

$$\frac{dv(t)}{dt} = r \Big/ \sum_{i\in B(t)} r_i , \tag{1}$$

where $B(t)$ is the set of sessions that are backlogged in the GPS server at time t, and r is the server rate. WFQ, therefore, achieves delay bound and fairness properties very close to those of GPS. However, since all N sessions can join or leave the set of backlogged sessions during a packet transmission time, the worst-case complexity of maintaining the system virtual time is O(N), which makes the algorithm not suitable for practical deployment in high-speed packet networks. The latency of WFQ is as follows,

$$\Theta_i^{(WFQ)} = \frac{L_i}{r_i} + \frac{L_{\max}}{r}, \quad (2)$$

where L_i and L_{max} are the length of the packet from session i and the maximum length of the packet in the system, respectively.

The fairness index of WFQ is as follows:

$$FI_{WFQ} = \max\left(\frac{L_i}{r_i} + \frac{L_{\max}}{r_j} + C_i,\ \frac{L_j}{r_j} + \frac{L_{\max}}{r_i} + C_j \right), \quad (3)$$

where $C_i = \min\left((N-1)\frac{L_{\max}}{r_i},\ \max_{1 \le k \le N} \frac{L_k}{r_k} \right)$.

There have been many efforts to reduce the complexity while maintaining the performance characteristics close to WFQ; examples include SCFQ, FFQ, SPFQ, and MD-SCFQ. SCFQ uses the virtual finish time of the packet that is currently being transmitted as the system virtual time. As a result, the complexity of computing the system virtual time of SCFQ is O(1), and it becomes more feasible for the high-speed network than for WFQ. It also has optimal fairness properties as follows:

$$FI_{(SCFQ)} = \frac{L_i}{r_i} + \frac{L_j}{r_j}. \quad (4)$$

However, the price of the easy system virtual time computation is a reduced level of isolation among the sessions, which causes the end-to-end delay bounds to grow linearly with the number of sessions that share the outgoing link. The delay bound of SCFQ with N sessions is given by

$$\Theta_i^{(SCFQ)} = \frac{L_i}{r_i} + (N-1)\frac{L_{\max}}{r}. \quad (5)$$

FFQ, SPFQ, and MD-SCFQ are all included in the rate proportional server (RPS) class [11]. As we can see by the basic properties of RPS shown in the following subsection, if a scheduling algorithm is included in the RPS class, then the algorithm has the same delay properties as those of WFQ. Therefore, the latencies of all these algorithms are the same as those of WFQ shown in (2). The fairness properties of each of them depend on how the system virtual time is recalibrated at every event. FFQ recalibrates the system virtual time when the session virtual times of all the backlogged sessions exceed the fixed value of frame time T. Therefore, the fairness bound is affected by the frame time T, while the complexity of the system virtual time computation is O(1). The fairness index of FFQ is

$$FI_{FFQ} = 2T + \max\left(\frac{L_i}{r_i}, \frac{L_j}{r_j} \right). \quad (6)$$

SPFQ derives the system virtual time from the minimum virtual start time of head-of-line (HOL) packets in all backlogged sessions. It has comparable fairness characteristics to WFQ. However, it needs another sorting structure to maintain the minimum virtual start time of HOL packets in all backlogged sessions. The resulting complexity of system virtual time computation is of O(log N). The fairness index of SPFQ is

$$FI_{SPFQ} = \max\left(\frac{L_i}{r_i}, \frac{L_j}{r_j} \right) + \max_{1 \le k \le N} \frac{L_k}{r_k} + \frac{L_{\max}}{r}. \quad (7)$$

MD-SCFQ recalibrates the system virtual time based on the weighted average virtual start time of all backlogged sessions. It also has fairness characteristics comparable to WFQ. However, the weighted average of the virtual start time of all backlogged sessions needs additional computation. The fairness index of MD-SCFQ is

$$FI_{MD-SCFQ} = \max(f_{i,j}, f_{j,i}), \quad (8)$$

where

$$f_{i,j} = \frac{L_i}{r_i} + \max\left(\frac{L_{\max}}{r_j}, \max_{1 \le k \le N} \frac{L_k}{r_k} - \frac{r_i}{r - r_j}\left(\max_{1 \le k \le N} \frac{L_k}{r_k} - \frac{L_i}{r_i} \right) - \frac{L_i}{r} \right).$$

The fairness indices, maximum delay bounds, and computation complexities of the above scheduling algorithms are compared in Table 1. In summary, most of the efforts to reduce the complexity of the virtual time computation deteriorate the system performance characteristics, such as delay and fairness properties, and most of the efforts to get good performance characteristics require additional complexities.

The purpose of this paper is to propose a new scheduling algorithm that has a low virtual time computation complexity along with good performance characteristics. Before we go through the proposed algorithm, in the following subsection, we will review the rate proportional server on which our algorithm is based.

2. Rate Proportional Server

The rate proportional server is a framework of scheduling algorithm that was formulated by Stiliadis and Varma in [11]. If a scheduling algorithm is included in the RPS category, then the delay bounds are the same as those of WFQ. Since our algorithm is based on the RPS, we need to review it here. The RPS is explained in terms of system virtual time, session virtual time, and service rules.

Table 1. Comparison of characteristics of fair queuing algorithms.

Algorithm	Fairness index	Latency	Virtual time computation complexity
WFQ	$\max\left(\frac{L_i}{r_i}+\frac{L_{\max}}{r_j}+C_i, \frac{L_j}{r_j}+\frac{L_{\max}}{r_i}+C_j\right)$, where $C_i=\min\left((N-1)\frac{L_{\max}}{r_i}, \max_{1\le n\le N}\frac{L_n}{r_n}\right)$	$\frac{L_i}{r_i}+\frac{L_{\max}}{r}$	O(N)
SCFQ	$\frac{L_i}{r_i}+\frac{L_j}{r_j}$	$\frac{L_i}{r_i}+(N-1)\frac{L_{\max}}{r}$	O(1)
FFQ	$2T+\max\left(\frac{L_i}{r_i}, \frac{L_j}{r_j}\right)$, where T is a frame time.	$\frac{L_i}{r_i}+\frac{L_{\max}}{r}$	O(1)
SPFQ	$\max\left(\frac{L_i}{r_i}, \frac{L_j}{r_j}\right)+\max_{1\le n\le N}\frac{L_n}{r_n}+\frac{L_{\max}}{r}$	$\frac{L_i}{r_i}+\frac{L_{\max}}{r}$	O(logN)
MD-SCFQ	$\max(f_{i,j}, f_{j,i})$, where $f_{i,j}=\frac{L_i}{r_i}+\max\left(\frac{L_{\max}}{r_j}, \max_{1\le n\le N}\frac{L_n}{r_n}-\frac{r_i}{r-r_j}(\max_{1\le n\le N}\frac{L_n}{r_n}-\frac{L_i}{r_i})-\frac{L_i}{r}\right)$	$\frac{L_i}{r_i}+\frac{L_{\max}}{r}$	O(1), but additional computation is required.

In an RPS, the session virtual time $v_i(t)$ must satisfy the following three properties.

① $v_i(t)$ is constant as long as session *i* is not backlogged.

② If session *i* becomes backlogged at time τ, then

$$v_i(\tau)=\max\left(v_i(\tau^-), v(\tau^-)\right), \tag{9}$$

where τ^- denotes the time instant just before the time τ.

③ For every time $t>\tau$ when session i remains backlogged, the session virtual time of the session increases by the normalized service amount offered to that session during the whole time interval, $(\tau, t]$,

$$v_i(t)=v_i(\tau)+\frac{W_i(\tau,t)}{r_i}. \tag{10}$$

In an RPS, the system virtual time of the fluid version of a scheduling algorithm must satisfy the following two properties.

① For any interval $(t_1, t_2]$ during a system busy period,

$$v(t_2)-v(t_1)\ge t_2-t_1. \tag{11}$$

② The system virtual time cannot exceed the minimum virtual time of all backlogged sessions at time *t*,

$$v(t)\le \min_{i\in B(t)} v_i(t). \tag{12}$$

At any time *t*, sessions are serviced according to the following rules.

① Among the backlogged sessions, only the set of sessions with the minimum session virtual time is serviced.

② Each session in this set is serviced with an instantaneous rate proportional to its reservation so as to increase the virtual times of the sessions in this set at the same rate.

SPFQ and MD-SCFQ satisfy the above RPS conditions. These algorithms follow the following basic rules to satisfy the RPS properties.

- The system virtual time increases linearly with time and is recalibrated at time instants $\tau_1, \tau_2, ..., \tau_k$. The time instants are called recalibration instants with the condition $\tau_1<\tau_2<...<\tau_k$, where τ_1 is the start time of the system busy period. In an actual system, the recalibration instants correspond to the packet departure event. That means that the system virtual time is recalibrated at every packet departure time.
- The system virtual time is recalibrated as follows at time instant τ_j :

$$v(\tau_j)=\max\left(v(\tau_j^-), SP(\tau_j)\right),$$

where $v(\tau_j^-)=v(\tau_{j-1})+\tau_j-\tau_{j-1}, v(\tau_0)=0$, and $SP(\tau_j)$ is any non-decreasing function that is specific to each different

scheduler.

• Basically the above rules are defined only in the system busy period. Therefore, when the system becomes idle, the system virtual time resets to zero. When the system busy period starts, the system virtual time is calculated according to the above rules.

III. A New Fair Scheduling Algorithm

Before we introduce the proposed scheduling algorithm, we make some assumptions and give an important definition in the algorithm. We have a finite number of different rates and a maximum packet length affordable in a system. When the maximum packet length is divided by a rate, we call the resulting value the maximum timestamp increment (MTI) of the rate:

$$MTI(R_j) = \frac{L_{\max}}{R_j}, \tag{13}$$

where R_j is a rate and $L_{\max}$ is the maximum packet length in a system.

The maximum MTI is a constant value that is determined at the system setup time. When the number of different rates in the system is D, the maximum MTI in a system can be defined as

$$MTI_{\max} = \max_{1 \le i \le D} \frac{L_{\max}}{R_j}. \tag{14}$$

The key idea of NSPFQ is that it recalibrates the system virtual time using the maximum MTI at the end of each packet transmission, while it uses the system virtual time for a newly arrived packet as the last calibrated system virtual time added by the elapsed real time between two calibration events.

In the following subsections we will explain the NSPFQ algorithm in detail and prove that NSPFQ has a rate-proportional property. Then, we will induce the fairness index by calculating the maximum difference of the normalized service received by two sessions during the continuously backlogged time interval.

1. The Service Discipline of NSPFQ

In NSPFQ, as in all GPS-related schedulers, the timestamps are assigned to the arriving packets according to the following equation:

$$F_i^k = \max\left(F_i^{k-1}, v(a_i^k)\right) + \frac{l_i^k}{r_i}, \tag{15}$$

where l_i^k is the length of the k-th packet from session i, and r_i is the reserved rate of session i. The system virtual time is maintained according to the same rule as other packet-by-packet rate proportional servers (PRPS) while the system virtual time at each recalibration time τ_i is calculated easily as shown below.

The system virtual time increases linearly with time at every packet arrival event and is recalibrated at every end of the packet departure time. The key point of the NSPFQ algorithm is the simple system virtual time recalibration method. We use the constant value to reduce the system virtual time computation complexity, which makes the NSPFQ algorithm affordable in a high-speed packet switching system. At every end of the packet transmission, a new packet with a minimum virtual finish time is selected from the HOL packets in all backlogged sessions. Since the virtual finish time (TS_{cur}) of the packet is the smallest value in the system at that time, if we subtract the maximum MTI from the TS_{cur}, the resulting value is the minimum possible value of the virtual start times of the HOL packets in the system at that time.

The detailed procedures of NSPFQ at the packet arrival time and at the end of the packet transmission are given below.

Procedures at the packet arrival time

① When a packet arrives at t, update the system virtual time using the recent system virtual time calculated at τ_{j-1}:

$$temp\_v(t) = v(\tau_{j-1}) + (t - \tau_{j-1}),$$

where $\tau_{j-1} \le t < \tau_j, \tau_0 = 0$ and $v(\tau_0) = 0$.

② Calculate the virtual finish time (i.e., the timestamp) of the packet:

$$F_i^k = S_i^k + l_i^k / r_i,$$

where $S_i^k = \max(F_i^{k-1}, temp\_v(t))$.

③ Place the packet with the timestamp in the related queue.

Procedures at the end of each packet transmission

① Increase the system virtual time by the transmission time of the packet just completed.

$$v(\tau_j^-) = v(\tau_{j-1}) + (\tau_j - \tau_{j-1}).$$

② Retrieve a packet with a minimum virtual finish time (TS_{cur}) from the HOL packets and transmit.

③ Recalibrate the system virtual time.

$$v(\tau_j) = \max(v(\tau_j^-), TS_{cur} - MTI_{\max}).$$

As we can see in (13), the maximum MTI is a constant value that is determined by the maximum packet size and the minimum session rate in the system. The NSPFQ algorithm uses the maximum MTI to calculate the minimum possible virtual start time. The minimum possible virtual start time at a

point is calculated only by a subtraction operation. When the maximum MTI is subtracted from the virtual finish time of a packet that is being transmitted, the result is the minimum possible value of the virtual start times of all the backlogged sessions. Therefore, our algorithm's system virtual time computation has a complexity of O(1). Additionally, NSPFQ requires a much smaller amount of information and less computation complexity than other GPS-related scheduling algorithms.

In the following section, we will show that our algorithm has good performance characteristics as well as this simplicity.

2. Performance Analysis of the NSPFQ Algorithm

In this section, we will show the performance characteristics of the NSPFQ algorithm. Since the RPS properties of the fluid version of a scheduling algorithm have an effect on the performance characteristics of scheduling algorithms, we prove that the fluid version of the NSPFQ algorithm is included in the RPS category. We then prove the delay and fairness properties of NSPFQ.

A. RPS Properties of the Fluid Version of the NSPFQ Algorithm

Before we prove that the fluid version of the NSPFQ algorithm is an RPS, we will use a lemma. The result of this lemma is the same as lemma 1 in [13]. However, we will prove this lemma with the definition of NSPFQ.

Lemma 1. Let τ_n and τ_{n+1} be two consecutive recalibration instants.

If $v_j(\tau_n) \ge v(\tau_n), \quad \forall j \in B(t),$

then $v_j(t) \ge v(t), \quad \forall j \in B(t), \quad \tau_n \le t \le \tau_{n+1}.$

Proof. From the definition of NSPFQ, the system virtual time increases by the real time passed. However, the session virtual time of a session increases by the amount of the service it received. Note that only sessions with minimum virtual time are serviced at any time.

Therefore, we can consider $v_j(\tau_n) \ge v(\tau_n), \forall j \in M(t)$ instead of $v_j(\tau_n) \ge v(\tau_n), \forall j \in B(t),$ where $M(t)$ is the set of sessions with the minimum session virtual time at time *t*:

$$M(t) = \left\{ j \middle| v_j(t) = \min_{k \in B(t)} v_k(t) \right\}.$$

The session virtual times with the minimum virtual time increase with the slope of $r / r_{M(t)}$ while the sessions leave set $M(t)$. Even though the set can change before the next recalibration time, the slope $r / r_{M(t)}$ is always larger than the system virtual time slope 1. With this condition, the system virtual time cannot exceed any session virtual time between the two recalibration times of the system virtual time. Therefore, we can conclude lemma 1. □

Lemma 2. The fluid version of NSPFQ is a rate proportional server.

Proof. For a fluid version of a packet scheduling algorithm to be classified as an RPS, the conditions of (11) and (12) must be satisfied. The former condition is met directly by the definition of the system virtual time of NSPFQ. Therefore, we only need to prove the latter condition and will prove it by induction, with reference to the recalibration instants as in [13].

Step 1. First we will show that (12) is satisfied from the start of the system busy period (τ_0) to the first recalibration time of the system virtual time (τ_1). That is, we should prove

$$v_j(t) \ge v(t), \quad \forall j \in B(t), \tau_0 \le t \le \tau_1,$$

where *B*(*t*) is the set of sessions that are backlogged in the system.

Since $v_j(\tau_0) = v(\tau_0) = 0, \forall j \in B(t),$ from lemma 1 it is clear that $v_j(t) \ge v(t), \ \forall j \in B(t), \ \tau_0 \le t < \tau_1.$

Now we have to show that it is still valid at time τ_1 when the first recalibration of the system virtual time occurs. Let $S(\tau_1)$ be the session whose first packet completes transmission at time τ_1. If we let the virtual finishing time of the transmitted packet be $F_{s(\tau_1)}$, it is also the session virtual time of $S(\tau_1)$ at that time.

Since session $S(\tau_1)$ was receiving service at time τ_1, $F_{s(\tau_1)}$ is the minimum among the virtual times of the backlogged sessions, which is also greater than the virtual start time of all HOL packets of any other sessions.

$$va_j \le F_{s(\tau_1)} \le v_j(\tau_1), \quad \forall j \in B(\tau_1^-) \setminus \{s(\tau_1)\},$$

where va_j is the virtual start time of session *j*.

From the definition of NSPFQ, the system virtual time increases linearly with the real time until τ_1 and is recalibrated to the maximum between the linearly increased time and the minimum value of all possible virtual start times at time τ_1. From lemma 1, we know that the linearly increased time cannot exceed the session virtual time. Therefore, the system virtual time is recalibrated to the minimum value of all possible virtual start times of HOL packets in backlogged sessions.

Therefore, $v_j(t) \ge v(t), \quad \forall j \in B(t), \tau_0 \le t \le \tau_1.$

Step 2. Now we will show that if (12) is satisfied until recalibration instant τ_{n-1}, then it is satisfied until recalibration instant τ_n (including τ_n). That is, we will prove the following.

$$v_j(\tau_{n-1}) \ge v(\tau_{n-1}) \ \ \forall j \in B(\tau_{n-1})$$
$$\Rightarrow v_j(t) \ge v(t) \ \ \forall j \in B(t), \tau_{n-1} < t \le \tau_n.$$

Since we know the result for $\tau_{n-1} \le t < \tau_n$ from lemma 1, we need to prove that $v_j(t) \ge v(t), \ \forall j \in B(\tau_n)$.

Now let $s(\tau_n)$ be the session whose packet completes transmission at time τ_n, and $F_{s(\tau_n)}$ be the virtual finishing time of the transmitted packet. As in step 1, since session $S(\tau_n)$ was receiving service at time τ_n, $F_{s(\tau_n)}$ is the minimum among the virtual times of backlogged sessions.

$$F_{s(\tau_n)} \le v_j(\tau_n), \quad \forall j \in B(\tau_n).$$

Even though virtual start times of the sessions that are backlogged at time τ_n can be greater than $F_{s(\tau_n)}$, they never exceed $v_j(\tau_n)$. If we note that NSPFQ recalibrates the system virtual time to the minimum virtual start time among all possible virtual start times, it is clear that $v(\tau_n) \le va_j(\tau_n) \le v_j(\tau_n)$. Therefore,

$$v_j(t) \ge v(t), \quad \forall j \in B(t), \quad \tau_{n-1} \le t \le \tau_n.$$

This concludes the proof through steps 1 and 2. □

B. Delay Properties of NSPFQ

The fluid version of NSPFQ presented above is an RPS. Therefore, the NSPFQ algorithm has all the delay properties of a PRPS, which is the same as those of WFQ. Consequently, from the delay properties of the PRPS scheduler, the latency of the NSPFQ scheduler is given as follows,

$$\Theta_i^{(NSPFQ)} = \frac{L_i}{r_i} + \frac{L_{\max}}{r}. \tag{16}$$

When a session i is constrained by the (σ_i, r_i) leaky bucket that has a reserved rate r_i and a burst size σ_i, the amount α_i of information units arriving at the server is bounded as

$$\alpha_i(\tau, t) \le \sigma_i + r_i(t - \tau) \tag{17}$$

during any time interval $(\tau, t]$ such that $\alpha_i(\tau, t) = \alpha_i(t) - \alpha_i(\tau)$.

For such a session, for an arbitrary network of NSPFQ servers, the maximum delay D_i^K after the K-th node in the network is bounded as

$$D_i^K \le \frac{\sigma_i}{r_i} + \sum_{j=1}^{K} \Theta_i^j - \frac{L_i}{r_i}. \tag{18}$$

C. Fairness Properties of NSPFQ

Since the recalibrations prevent the system virtual time from lagging indefinitely behind the virtual times of the sessions that are currently backlogged in a PRPS system and the NSPFQ algorithm is a PRPS, we can say that NSPFQ is a fair scheduler. However, we need to define the formal fairness index. We adopt Golestani's definition of the fairness index [10]. The fairness index is defined as

$$\left| \frac{W_i(t_1, t_2)}{r_i} - \frac{W_j(t_1, t_2)}{r_j} \right| \le F_{i,j}, \tag{19}$$

where $W_i(t_1, t_2)/r_i$ is the normalized service received by session i during the continuously backlogged time interval (t_1, t_2).

Lemma 3. In a PRPS system, at any time t, the difference between the system virtual time $v(t)$ and the timestamp F_i^k of any packet P_i^k of session i that is currently in the system is bounded as follows:

$$v(t) - F_i^k \le \frac{L_{\max}}{r_i}, \quad \forall i \in B(t),$$

where r_i is the reserved rate of session i.

Proof. For any PRPS and corresponding RPS, the following holds [12]:

$$t_i^k \le \hat{t}_i^k + \frac{L_{\max}}{r}, \tag{20}$$

where t_i^k is the time when a packet completes transmission in the considered PRPS, and $\hat{t}_i^k$ is the time when the same packet p_i^k completes transmission in the corresponding RPS.

Assume the n-th recalibration of system virtual time τ_n occurs at time t_i^k.

The session virtual time $v_i(t_i^k)$ is as follows:

$$v_i(t_i^k) = v_i(\hat{t}_i^k) + \frac{r}{r_{M(t)}}(t_i^k - \hat{t}_i^k), \tag{21}$$

where $M(t) = \left\{ j \middle| v_j(t) = \min_{k \in B(t)} v_k(t) \right\}$.

In an RPS system, when packet p_i^k completes transmission in the fluid system, its timestamp F_i^k is greater than or equal to the system virtual time:

$$v(\hat{t}_i^k) \le F_i^k. \tag{22}$$

Therefore, according to (21) and (22) with the second condition of the RPS system, $v(t) \le \min_{i \in B(t)} v_i(t)$, the system virtual time at time t_i^k is bounded as follows,

$$\begin{aligned} v(t_i^k) &\le v(\hat{t}_i^k) + \frac{r}{r_i}(t_i^k - \hat{t}_i^k) \\ &\le F_i^k + \frac{L_{\max}}{r_i}. \end{aligned} \tag{23}$$

This concludes the proof of lemma 3. □

Since the system virtual time between the two recalibration times increases by the real time elapsed, the following corollary comes directly.

Corollary 1. In a PRPS system, at time s_i^k, when packet p_i^k starts transmission, the difference between the system virtual time and its timestamp F_i^k is

$$v(s_i^k) - F_i^k \le \frac{L_{\max}}{r_i} - \frac{l_i^k}{r}. \tag{24}$$

We can evaluate the fairness index (FI) of NSPFQ following the procedures presented in [13].

Theorem 1. The fairness index of NSPFQ is

$$FI_{(NSPFQ)} = \max(f_{i,j}, f_{j,i}),$$

where $f_{i,j} = \frac{L_i}{r_i} + \max(\frac{L_j + L_{\max}}{r_j} - \frac{L_j}{r}, \max_{1 \le n \le D} \frac{L_{\max}}{R_n} - \frac{L_i}{r})$.

Proof. We can derive $FI_{(NSPFQ)}$ from the difference of normalized service received by session *i* and *j* at any time interval, during (t_1, t_2) when the two sessions are continuously backlogged. All the possible cases that we can consider are as follows.

1. One session becomes backlogged at time t_1, while the other session is already backlogged before time t_1.
2. Both sessions become backlogged at time t_1.

Even though we can consider one more cases where both sessions are backlogged before time t_1, it can be treated as case 1 in any time $t' < t_1$.

Case 1. Session *j* becomes backlogged at time t_1, whereas session *i* is already backlogged at time t_1, and F_i^k is the timestamp of its HOL queue packet p_i^k. We must consider two subcases.

Subcase 1.1. Packet p_j^m of session *j* receives a timestamp $F_j^m > F_i^k$.

In order to maximize the amount of normalized service provided to session *i* before session *j* receives its first service, packet p_j^m must reach the system at the exact time when the packet p_i^k is picked for transmission.

According to lemma 3 and corollary 1, at the exact time satisfying the condition, the virtual finishing time of session *j*, F_j^m is

$$F_j^m \le F_i^k + \frac{L_{\max}}{r_i} - \frac{l_i^k}{r} + \frac{l_j^m}{r_j}, \tag{25}$$

where l_i^k and l_j^m are the lengths of packet p_i^k and p_j^m, respectively.

The service provided to packet p_i^k (equal to l_i^k / r_i) contributes to the difference of normalized service, and all the following packets of session *i* having a timestamp not greater than F_j^m are transmitted before p_j^m.

From (25)

$$\begin{aligned} F_j^m - S_i^k &= F_j^m - (F_i^k - \frac{l_i^k}{r_i}) \\ &\le \frac{L_{\max}}{r_i} + \frac{l_j^m}{r_j} - \frac{l_i^k}{r} + \frac{l_i^k}{r_i}, \end{aligned}$$

where S_i^k is the virtual start time of packet p_i^k.

The resulting difference of received normalized service before packet p_j^m is picked for transmission is, therefore, bounded by

$$f_{j,i}^{(1a)} = \frac{L_i + L_{\max}}{r_i} - \frac{L_i}{r} + \frac{L_j}{r_j}. \tag{26}$$

Subcase 1.2. Packet p_j^m of session *j* receives a timestamp $F_j^m < F_i^k$.

In order to maximize the amount of normalized service provided to session *j* before session *i* receives service again, packet p_j^m must reach the system just before packet p_i^k completes transmission and the system virtual time is consequently recalibrated. Let l_i^k be the length of the packet p_i^k, and l_i^{k+1} be the length of p_i^{k+1}. Packet p_i^k started transmission at time $t_s = t_1 - l_i^k / r$; at that time, according to the definition of NSPFQ, the minimum possible value of the system virtual time is

$$\begin{aligned} v(t_s) &= F_i^k - MTI_{\max} \\ &= F_i^k - \max_{1 \le n \le D} \frac{L_{\max}}{R_n}. \end{aligned} \tag{27}$$

Packet p_i^{k+1} receives a timestamp $F_i^{k+1} = F_i^k + l_i^{k+1} / r_i$.

If p_j^m reaches the system at time t_1 just before p_i^k completes transmission, the minimum possible value of the system virtual time is

$$v(t_1) = v(t_s) + \frac{l_i^k}{r}, \tag{28}$$

so that the resulting difference of normalized service between session *j* and session *i* can be expressed as

$$F_i^{k+1} - v(t_1) = \frac{l_i^{k+1}}{r_i} + \max_{1 \le n \le D} \frac{L_{\max}}{R_n} - \frac{l_i^k}{r}. \tag{29}$$

The whole difference of normalized service in (29) increases

with l_i^k and l_i^{k+1}, so that

$$f_{i,j}^{(1b)} = \frac{L_i}{r_i} + \max_{1 \le n \le D} \frac{L_{\max}}{R_n} - \frac{L_i}{r}. \quad (30)$$

Case 2. Both sessions *i* and *j* become backlogged simultaneously at time t_1.

In this case, both sessions use the same system virtual time to calculate the virtual finish times. Therefore, the maximum difference of the received normalized service between them is decided by their virtual finish times. Without loss of generality, assuming that $l_i^k / r_i > l_j^k / r_j$, session *i* begins service before session *j*. Consequently, the maximum difference of received normalized service between them at any time t_2, when p_i^k ends the transmission, is

$$f_{i,j}^{(2)} \le \frac{L_i}{r_i}. \quad (31)$$

Therefore, theorem 1 follows from (26), (29) and (31). □

3. Extension of the NSPFQ Algorithm

We introduced a simple scheduling algorithm that has a lower computation complexity for the system virtual time. This simple mechanism can be extended to get better fairness on a statistical basis making its fairness index unchanged.

Instead of the using one maximum MTI determined at the system setup time, we can use the maximum of the MTIs of all backlogged sessions. In this case, we need another sorting structure to maintain the maximum MTI among all the backlogged sessions. However, we can restrict the number of MTIs in the system to improve the system performance. In this case, we have two options. First, we can support only a fixed number of rates, that is, we only support the designated rates without supporting the other rates. This makes the implementation easy, but introduces some limitation in the operation of the system. Second, we can support many rates. Only MTIs for the fixed number of representative rates are calculated and the MTIs for the other rates are taken from that of the nearest rate.

The sorting structure can be implemented with a max heap, in which the elements are different values of MTIs and each element keeps the number of sessions having their MTIs. Refer to the following section for some information on the heap. When a session is newly backlogged in the system, we can have two cases in the binary search tree insertion operation. First, when the MTI does not exist in the heap, the timestamp increment is inserted in the heap. Second, when the MTI exists, the number for the MTI is incremented. When a session goes out of the backlogged status, the number of the MTI for the session is decremented. If the number of the MTI becomes zero as a result of the decrement operation, the MTI is deleted in the heap. When we select the maximum MTI at the end of every packet transmission event, we just need to read the root of the heap.

This operation could be implemented easily in the high-speed network without much complexity. The complexity of maintaining this additional sorting structure is O(log*D*), where *D* is the number of rates supported in the system. Considering that the complexity of maintaining the virtual start time is O(log*N*), where *N* is the number of the backlogged sessions, the NSPFQ algorithm has much lower complexity than SPFQ even with this extended method.

IV. Hardware Implementation Framework

In this section we propose a hardware implementation framework for the NSPFQ scheduling algorithm as shown in Fig. 1. In the following subsections, we will explain the overall components and then provide the details of the heap manager architecture.

1. Proposed Scheduler Architecture

The proposed scheduler is composed of a POS-PHY III interface, NSPFQ scheduler, heap manager, shaping controller, VOQ controller, port scheduler, and PCI interface module. We assume that the queue management function works in another module called the queue manger (QM) and the scheduler gets the packets from that module. The explanation for each module is as follows.

- POS-PHY III input data interface

This module takes care of the interface with the QM. The QM sends packets with information such as the flow ID, destination port, service rate, and service priority. It works as a POS-PHY III slave.

- PCI bus controller

The PCI bus is for the host CPU interface, which is used to initialize the scheduler at the startup time and to change the configuration during the run time.

- NSPFQ scheduler

NSPFQ gets packets through the POS-PHY III input data interface with information such as destination port, service priority, etc. Then, the virtual finish time is calculated as in the method in section III. After the virtual finish time is calculated, packets are sorted in the heap manager according to the destination port and service priority.

- Heap manager

The number of heap manages in the system is decided by the number of ports multiplied by the number of classes. Each heap manager sorts the packets according to their virtual finish time so that the packet with the minimum virtual finish time is served first.

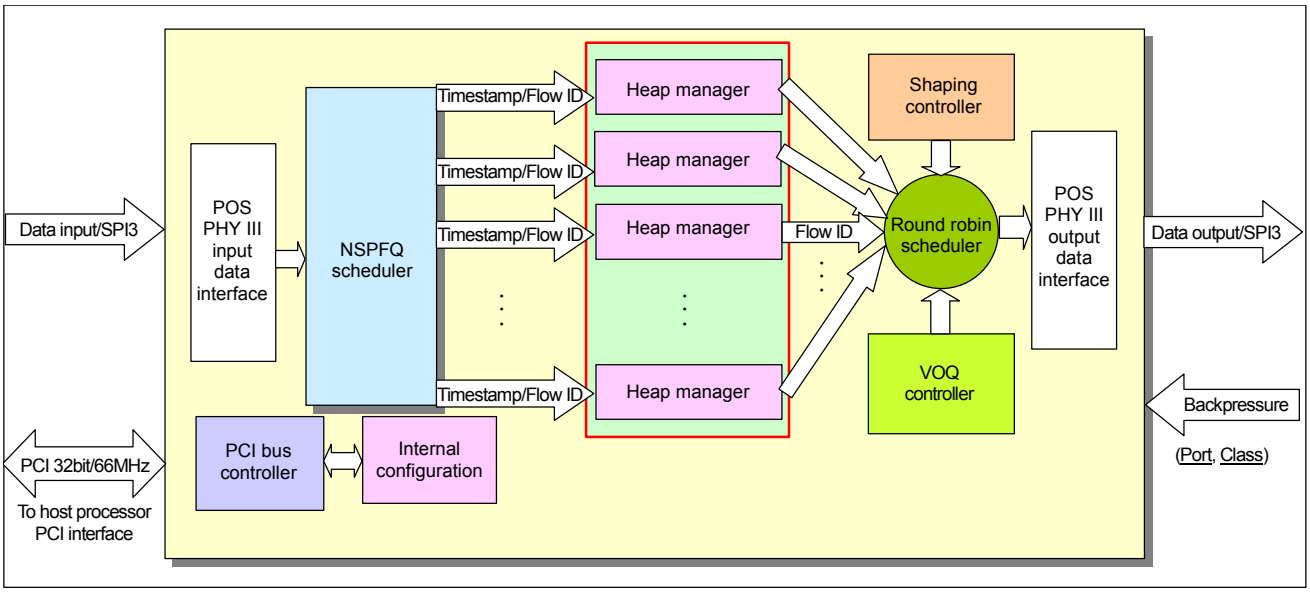

Fig. 1. An example of a scheduler using the NSPFQ algorithm.

- Round robin SCAN

The round robin SCAN schedules the VOQ. As the name indicates, the ports are serviced in a round robin manner.

- Shaping controller

The shaping controller regulates the traffic burstiness by controlling the inter-packet gaps.

- VOQ controller

The VOQ controller gets the backpressure information from the backpressure interface. Then, it prevents the VOQ from sending packets.

- POS-PHY III output data interface

This module takes care of the interface with the QM. The QM sends packets with information such as the flow ID, destination port, service rate, and service priority. It works as a POS-PHY III slave.

- Backpressure interface

The backpressure interface receives the backpressure information from the QM.

The core of the scheduler is the scheduling algorithm; it has been sufficiently explained in the previous sections. We focus now on the sorted priority queues among the above modules.

2. Heap Manager

A lot of sorted priority queues have been studied in the literature, such as the binary tree of the comparator-based priority queue, shift register-based priority queue, systolic array-based priority queue, calendar queue, and heap manager [17]. The binary tree of the comparator-based priority queue, shift register-based priority queue, and systolicarray-based priority queue are not scalable. Calendar queue and heap manager are used widely.

The calendar queue has a O(1) complexity, but it needs an additional tree structure to find an empty slot, so it actually has a O(log$N$) complexity. Furthermore, it is not that scalable considering the scheduling according to the port and service class. As shown in the next equation, the number of queues of the calendar queue depends on the service rates that it supports.

$$\text{Number of Queues} = \frac{\text{Maximum service rate of flow}}{\text{Minimum service rate of flow}}$$

For example, since it supports 10 kbps minimum service rate and 10 Gbps maximum service rate, the number of queues needed is 100 M. Additionally, the calendar queue has to exist per port and class. Therefore, the total memory requirement for the above example is "100M×number of ports×number of classes×memory requirement for a queue."

Heap is a complete binary tree in which each node has a unique key. The node of the maximum value or the minimum value can be found easily in the data structure. Heap can be implemented by the two methods, max heap and min heap. Each node in max heap has a key that is less than or equal to the key of its parent and each node in min heap has a key that is greater than or equal to the key of its parent. Since we need to find the packets with the minimum virtual finish time, we use min heap. Heap has O(N) complexity. However, when it is pipelined, it has O(log N) complexity. Furthermore, since the memory requirement only depends on the number of flows, it is more scalable than the calendar queue and it supports any

service rate.

The heap manager is composed of a 16-heap data structure module (per port heap manager) and a class multiplexing module (Fig. 2). Each heap data structure module has a memory module and controller per level, a non-empty counter & register module, and a node status register & register module. The class multiplexing module multiplexes timestamps according to their classes to insert into the heap manager per port. A heap data structure module per switch port exists and has queues for 256 flows. Each level of memory is composed of DPRAM. The memory controller controls the DPRAM read/write. The non-empty counter & register module finds a valid path for the enqueue operation. The node status register & controller is used as a flag to indicate that each node has a flow to service. The controller sends the ACK for a service flow to the NSPFQ scheduler. Each node of DPRAM stores 58 bits of information. Each heap manager receives packets and information related to the NSPFQ and services the flows according to their service priority.

V. Simulation Results

In this section we present the simulation results to verify the performance of NSPFQ. Although we have analyzed the upper bound on delays of the NSPFQ algorithm, it is important to compare the actual delays seen by sessions in a realistic network configuration. We compared the proposed algorithm with WFQ, SCFQ, and SPFQ. We also traced the session virtual times for all the sessions to show that the fairness for the NSPFQ is good and almost identical to the other fair queuing algorithms indirectly.

1. Simulation Model and Traffic Source

We simulate the algorithms in a single node configuration. Eight sessions share the same outgoing link and session 1 is

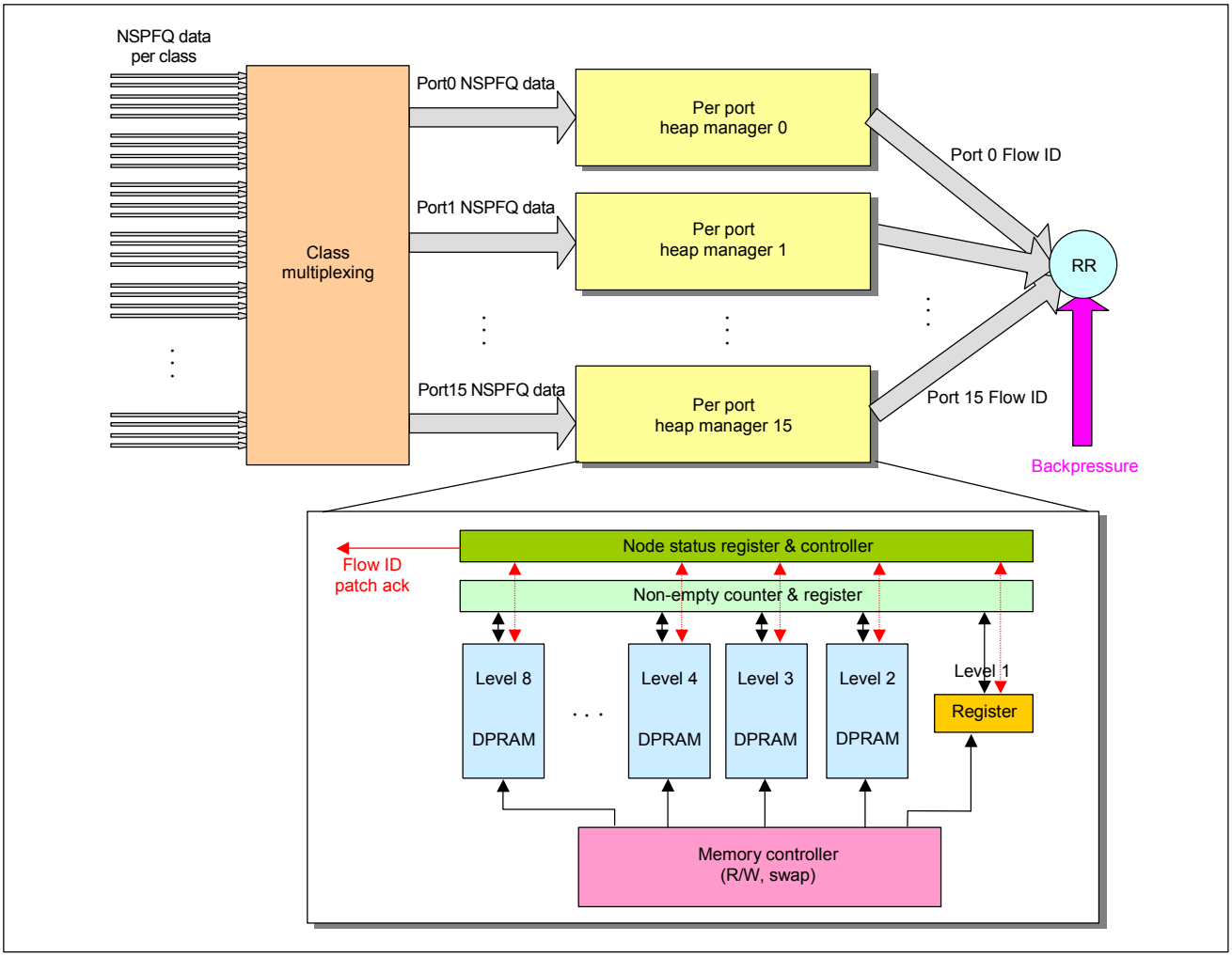

Fig. 2. Details of the heap manager.

misbehaving while others are transmitting within their reservations. There is no blocking at the node and the scheduler is assumed to have an infinite buffer capacity for queuing packets.

An ON-OFF model is used in the simulations to investigate the scheduler performance. The packet stream consists of arrivals with T ms intervals when the model is in an ON state. We choose one packet time as T. There are no arrivals when the model is in an OFF state. Packet arrivals during the ON state are approximated by a geometric distribution with mean $\lceil 1/(\alpha T) \rceil$. The duration of the OFF state is distributed exponentially with means $1/\beta$.

We set $1/\alpha$ to $100\, r_i$ and $1/\beta$ to $100\,(1-r_i)$, where r_i is the reservation of session *i*. A well-behaving session is shaped by the (σ_i, r_i) leaky bucket, where σ_i and r_i are the burst size and rate of session *i*, respectively.

2. Simulation Results

A. Delay Properties

We selected the burst size $\sigma_i = 2$ for each session. We also assumed that session 1 is misbehaving and attempting to transmit more than its reservation. Tables 2 and 3 show the average delay and maximum delay of algorithms in terms of the one fixed-packet transmission time, respectively. The average and maximum delay seen by session 0, which has 50% of the link bandwidth, is substantially higher in the SCFQ server than other servers. WFQ, SPFQ, and NSPFQ provide the same average and maximum delay. It is easy to verify that NSPFQ has almost the same average and maximum delay performances as those of SPFQ.

B. Fairness Properties

Figures 3, 4, and 5 show the trace of the session virtual times of the WFQ, SPFQ, and NPSFQ algorithms with the same simulation environment for the delay properties. This could be one of ways to show the fairness properties of scheduling algorithms. The session virtual time trace of NSPFQ is almost identical to those of the WFQ and SPFQ. In addition, the session virtual times of sessions are going along the same path without diverging.

Figures 6, 7, and 8 show traces of normalized service with the same simulation environment for the delay properties. These results show that WFQ, SPFQ, and NSPFQ have almost the same fairness property as in the above session virtual time trace. Even though we can see few differences in normalized services among the sessions in the middle of the each figure, they came from the traffic properties at that time period.

Table 2. Average delay.

	Reserved rate	Arrival rate	WFQ	SCFQ	SPFQ	NSPFQ
0	0.500000	0.498	1.5913	3.08040	1.5917	1.5944
1	0.062500	0.100	N/A	N/A	N/A	N/A
2	0.062500	0.062	8.5502	14.5631	5.4680	4.2446
3	0.062500	0.061	8.9832	14.4902	5.7585	4.4188
4	0.078125	0.076	6.3890	11.9415	3.8515	3.0688
5	0.078125	0.076	6.4524	11.7837	3.5661	3.0229
6	0.078125	0.076	5.8597	11.6815	3.2028	2.6831
7	0.078125	0.076	6.5652	11.8675	4.1405	3.3338

Table 3. Maximum delay.

	Reserved rate	Arrival rate	WFQ	SCFQ	SPFQ	NSPFQ
0	0.500000	0.498	2.0000	7.6000	5.0000	5.0000
1	0.062500	0.100	N/A	N/A	N/A	N/A
2	0.062500	0.062	26.0000	31.6000	26.0000	26.0000
3	0.062500	0.061	28.0000	26.6000	28.0000	28.0000
4	0.078125	0.076	19.6000	25.9984	18.6000	18.5999
5	0.078125	0.076	19.6000	24.6000	19.4000	19.4000
6	0.078125	0.076	21.4000	23.9984	21.4000	21.4000
7	0.078125	0.076	22.4000	24.6000	22.4000	22.4000

VI. Conclusions

In this paper, we proposed a new fair queuing algorithm, called new starting potential fair queuing (NSPFQ). The NSPFQ algorithm is a simple scheduling algorithm that is similar to the SPFQ. However, it maintains the same sorting structure for the virtual start time as the SPFQ, while its performances are nearly identical to WFQ. Since NSPFQ belongs to the PRPS, we obtained its delay bounds, which are the same as those of WFQ. We also analyzed the fairness property of the algorithm and showed that the difference in normalized service offered to any two sessions that are continuously backlogged is always bounded and this bound is comparable to that of WFQ. We also provided an extended method that provides better fairness on a statistical basis and maintains the same fairness index.

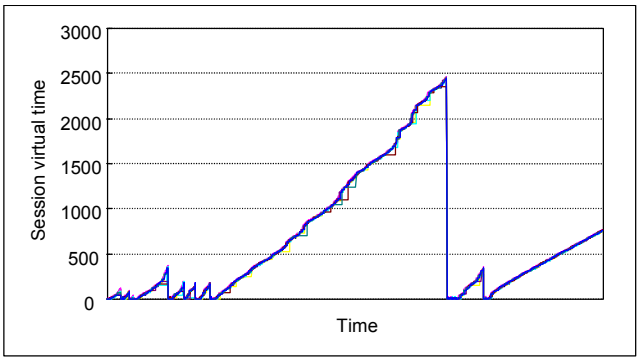

Fig. 3. Session virtual time trace of WFQ algorithm.

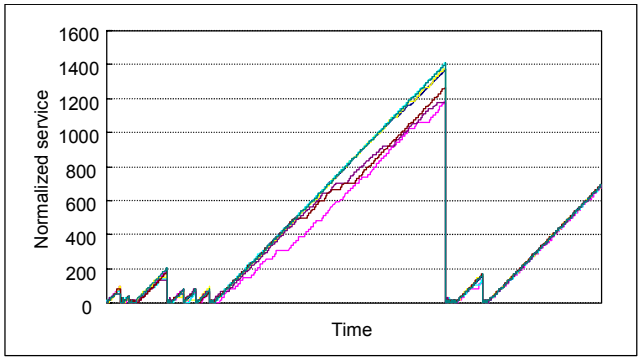

Fig. 6. Normalized service trace of WFQ algorithm.

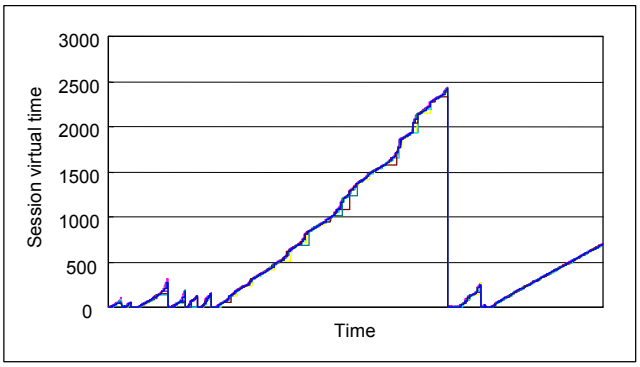

Fig. 4. Session virtual time trace of SPFQ algorithm.

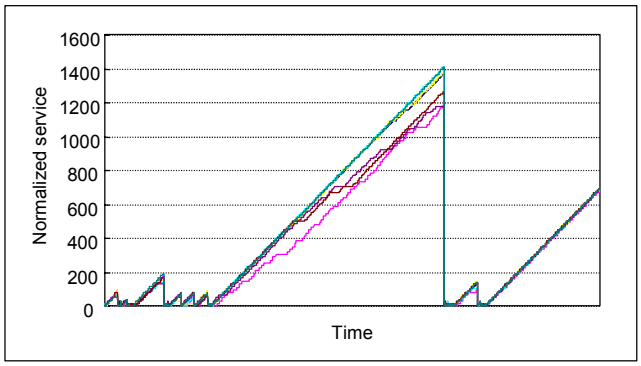

Fig. 7. Normalized service trace of SPFQ algorithm.

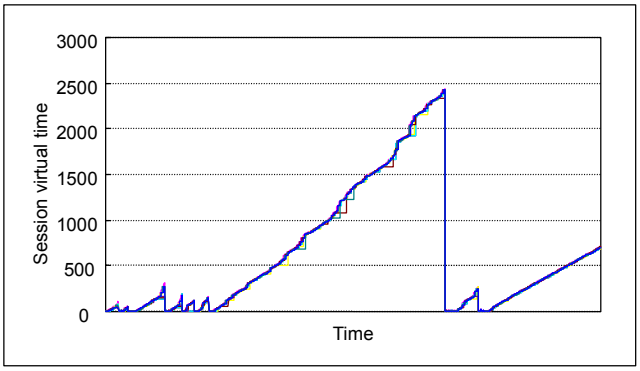

Fig. 5. Session virtual time trace of NSPFQ algorithm.

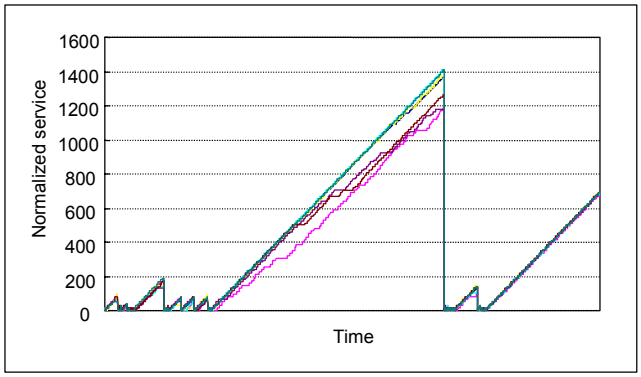

Fig. 8. Normalized service trace of NSPFQ algorithm.

References

[1] H. Zhang, "Service Disciplines for Guaranteed Performance Service in Packet Switching Networks," *Proc. of the IEEE*, vol. 83, no. 10, Oct. 1995, pp. 1374-1396.

[2] L. Zhang, "Virtual Clock: A New Traffic Control Algorithm for Packet Switching," *ACM Trans. on Computer Systems*, vol. 9, no. 2, May 1991, pp. 101-124.

[3] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis, "Weighted round-robin cell multiplexing in a general-purpose ATM switch chip," *IEEE JSAC*, vol. 9, Oct. 1991, pp. 1265-1279.

[4] M. Shreedhar and G. Varghese, "Efficient Fair Queuing Using deficit round robin," *Proc. of ACM SIGCOMM'95*, Sept. 1995, pp. 231-242.

[5] D. Verma, H. Zhang, and D. Ferrari, "Guaranteeing Delay Jitter Bounds in Packet Switching Networks," *Proc. of Tricomm'91*, Apr. 1991, pp. 35-46.

[6] C. Kalmanek, S. Morgan, and R.C. Restrick, "Rate Controlled Servers for Very High-Speed Networks," *Proc. of IEEE GLOBECOM*, Dec. 1990, pp. 300.3.1-300.3.9.

[7] S. Golestani, "A Framing Strategy for Congestion Management," *IEEE JSAC*, vol. 9, Sept. 1991, pp. 1064-1077.

[8] H. Zhang and S. Keshav, "Comparison of Rate-Based Service Disciplines," *Proc. of ACM SIGCOMM'91*, 1991, pp.113-122.

[9] A.K. Parekh and R.G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case," *Proc. of IEEE INFOCOM'92*, vol. 2, May 1992, pp. 915-924.
[10] S.J. Golestani, "A Self-Clocked Fair Queuing Scheme for Broadband Applications," *Proc. of IEEE INFOCOM '94*, Apr. 1994, pp. 636-646.
[11] D. Stiliadis and A. Varma, "Efficient Fair Queuing Algorithms for Packet-Switched Networks," *IEEE/ACM Trans. Networking*, vol. 6, no. 2, Apr. 1998, pp. 175-185.
[12] D. Stiliadis and A. Varma, "A General Methodology for Designing Efficient Traffic Scheduling and Shaping Algorithms," *Proc. of IEEE INFOCOM '97*, vol. 1, Apr. 1997, pp. 326-335.
[13] F.M. Chiussi and A. Francini, "Minimum-Delay Self-Clocked Fair Queuing Algorithm for Packet-Switched Networks," *Proc. of IEEE INFOCOM '98*, vol. 3, Mar. 1998, pp. 1112-1121.
[14] F.M. Chiussi et al., "A Family of ASIC Devices for Next Generation Distributed Packet Switches with QoS Supports for IP and ATM," *Hot Interconnects* 9, Aug. 2001, pp. 145-149.
[15] C. Dovrolis and D. Stiliadis, "Proportional Differentiated Services: Delay Differentiation and Packet Scheduling," *IEEE/ACM Trans. Networking*, vol. 10, no. 1, Feb. 2002, pp. 12-26.
[16] Byung-Hwan Choi and Hong-Shik Park, "Rate Proportional SCFQ (RP-SCFQ) Algorithm for High-Speed Packet-Switched Networks," *ETRI J.*, vol. 22, no. 3, Sept. 2000, pp. 1-9.
[17] Aggelos D. Ioannou, "An ASIC Core for Pipelined Heap Management to Support Scheduling in High Speed Networks," *FORTH-ICS / TR-278*, Nov. 2000.

Dong-Yong Kwak received his BS degree and MS degree both in computer science from Dongguk University in Seoul, Korea, in 1983 and 1985. He joined ETRI in 1985. From 1985 to 1990, he was on the technical staff of the software division developing TDX test environments. He was involved in an ATM switch development project from 1991 to 2001, and he is currently a leader in the Network Processor Technology Team. His research interests are in traffic scheduling algorithms, traffic engineering, and network processor design.

Nam-Seok Ko received his BS degree in computer engineering from Chonbuk National University in 1998 and MS degree from Information and Communications University (ICU) in 2000. He is currently working for the Electronics and Telecommunications Research Institute in Daejeon, Korea. His research interests include high-speed network architecture, protocol, and security.

Bongtae Kim received his BS degree in electronics from Seoul National University, Seoul, Korea, in 1983, and his MS degree and PhD degrees both in computer engineering from North Carolina State University, Raleigh, NC, USA, in 1991 and 1995. He joined ETRI, Daejeon, Korea, in 1983. From 1983 to 1990, he was on the technical staff of the switching division, developing TDX switches. In 1985 and 1986, he was a Visiting Engineer of Network Systems, ITT Telecom, Raleigh, NC, USA, where he was involved in developing a digital concentrator. In 1996, he became the Team Leader of developing broadband multimedia communication services and service architecture for the nationwide ATM testbed in Korea. During 1997 and 1998, he worked as the Project Leader of developing QAM VDSL chips, and during 1999 and 2000, he worked as the Team Leader of ACE2000 ATM switch architecture design. In 2001, he assumed his current responsibility as the Director of the Network Core Technology Department of ETRI. His research interests are in network SoC and network system design, communication protocols, and queueing systems.

Hong-Shik Park received the BS degree from Seoul National University, Seoul, Korea in 1977, and the MS and PhD degrees from KAIST, Daejeon, Korea in electrical engineering in 1986 and 1995. In 1977, he joined the Electronics and Telecommunications Research Institute and had been engaged in development of TDX digital switching system family including TDX-1, TDX-1A, TDX-1B, TDX-10, and ATM switching systems. In 1998, he moved to ICU, Daejeon, Korea as a faculty member. Currently he is an Associate Professor. His research interests are network architecture, network protocols, and performance analysis of telecommunication systems. He is a member of the IEEK, KICS, Korea, and IEICE, Japan.