ELSEVIER

Computer Communications 23 (2000) 45–61

computer communications

www.elsevier.com/locate/comcom

Bounded tag fair queueing for broadband packet switching networks

Yen-Jen Chen*, S.-Y. Lee

Dept. of Computer Science and Information Engineering, National Chiao Tung University, Hsin-Chu, Taiwan, ROC

Received 11 September 1998; received in revised form 15 June 1999; accepted 29 June 1999

Abstract

Fair Queueing (FQ) is an attractive packet scheduling mechanism, which can establish firewall among packet flows. Thus, the quality of service (QoS) of each flow can be guaranteed. To implement the mechanism, three important performance issues are concerned: fairness, bounded delay, and efficiency. We propose a scheme, named BTFQ, which satisfies the three criteria. The scheme BTFQ + is also proposed to improve the fairness property of BTFQ and is designed to combine with a low-cost, high-speed hardware for reducing computation time. For comparison, we classify some well-known packet scheduling schemes into six classes according to their fairness properties. BTFQ and BTFQ + are excellent in bounded delay and efficiency. Their fairness properties, although a little bit weak, are but still very good if the traffic load is under 95%, according to the simulation results. Because of the tradeoff among the performance issues, BTFQ and BTFQ + provide another good choices. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Packet scheduling; Fair queueing; Fairness; Bounded delay; Efficiency

1. Introduction

In the high-speed network, various applications are integrated together, such as Video On Demand, Video Conference, Distant Learning, and Digital Telephony [1,2]. The packet flow, which is a sequence of packets generated from an application and travels along a fixed route, requires stringent performance in delay, packet loss, and throughput. One of the most important strategies to guarantee the performance is to fairly allocate the bandwidth of the network channel to the flows passing through it. This leads to good isolation for a flow without the interference from others.

Fair queueing (FQ) is a packet scheduling mechanism, which is capable of allocating the channel bandwidth fairly. The implementation of the mechanism concerns three important performance issues: fairness (in bandwidth allocation), bounded delay (for each packet), and efficiency (of scheduling algorithm) [4,14,15].

A lot of FQ schemes, offering fairness allocation to the flows sharing the channel, have been proposed such as Packet Generalized Processor Sharing (PGPS) [4,5] (also known as Weighted Fair Queueing [6]), Self-Clock Fair Queueing (SCFQ) [8], Start-time Fair Queueing (SFQ) [11], Leap Forward Virtual Clock (LFVC) [12], Minimum Starting-tag Fair Queueing (MSFQ) [13], Time Shift Scheduling (TSS) [14], Frame-based Fair Queueing (FFQ) [15,20,21], and Starting Potential-based Fair Queueing (SPFQ) [20,21]. They all can provide bounded delay if the traffics of the flows constrained by Leaky Bucket [16]. It is complex to implement PGPS in the high-speed network since PGPS needs to simulate the theoretic FQ scheme, Generalized Processor Sharing (GPS) [4], in real time. SCFQ and SFQ are easy to implement, but they both provide the delay bounds that grow with the number of the flows sharing the channel. LFVC, MSFQ, SPFQ, and TSS provide the small delay bounds independent of the number of flows while they are less efficient than SCFQ. FFQ is as efficient as SCFQ and provides the same delay bound as TSS, but its fairness property is worse than those of the above schemes.

Here we propose an efficient FQ scheme, called Bounded Tag Fair Queueing (BTFQ), which can allocate bandwidth fairly and offer a small delay bound, independent of the number of flows. We also propose the enhancement version, BTFQ + , which is an improvement on BTFQ in fairness. In addition, hardware is designed for BTFQ + to reduce the time on packet scheduling. The time that the hardware requires for scheduling a packet is less than the time required by an ADD operation in a 32-bit adder [17], if the number of the flows sharing the channel is less than 4×10^9.

The remaining of the paper is organized as follows. In Section 2, we introduce the goal of FQ and the formal

* Corresponding author. Tel.: + 886-35712121; fax: + 886-35724176.
E-mail address: ctchen@csie.nctu.edu.tw (Yen-Jen Chen)

Nomenclature

N	number of maximum flows allowed; flow identifier is in the range $[0,N-1]$
$B(t1,t2)$:	set of the backlogged flows during $[t1, t2]$
$B(t)$:	$\lim_{\Delta t\to 0} B(t,t+\Delta t)$
C:	server capacity in bits per second (bps)
$W.f(t1,t2)$:	data amount of flow f (in bits) sent by server during $[t1,t2]$
$G.f(t1,t2)$:	bandwidth of flow f during $[t1,t2]$, which is equal to $W.f(t1,t2)/(t2-t1)$
$R.f$:	reserved rate for flow f in bps
$F.ij(t1,t2)$:	fairness between any two flows, i and j, backlogged during $[t1,t2]$
$D.p$:	delay of packet p passing through the server (including queueing delay)
queue.f:	queue for flow f
$L.p$:	length (in bits) of packet p
$L.f$:	length (in bits) of the packet at the head of queue.f
$L.f_{\max}$:	upper bound on packet length for flow f
$L_{\max}$:	upper bound on packet length for all flows
$S_{\max}$:	$\max_{i=0\ldots N-1}\{L.i_{\max}/R.i\}$.
$S_{\max+}$:	maximum of $L.i_{\max}/R.i$ of the flows i currently backlogged and served
$X.f$:	priority of queue.f
$X.p$:	priority of packet p, which is the queue priority when p is at the head of the queue
XS:	priority of the packet currently served
H:	ordered list of the queue priorities of all backlogged flows
$T.f$:	tag of queue.f, an auxiliary for computing $X.f$
$T.p$:	tag of packet p, which is the queue tag when p is at the head of the queue
T_{bnd}:	lower bound of the queue tags of all backlogged flows
$V.p$:	virtual arrival time of packet p
TagClock:	special clock helping determine $V.p$ in the schemes BTFQ and BTFQ +

definitions of fairness, bounded delay, and efficiency. In Section 3, we describe BTFQ and show its performance. In Section 4, the enhanced scheme BTFQ + is proposed and the architecture of the supporting hardware is illustrated. In Section 5, we classify the famous scheme, Virtual Clock (VC) [3], and some FQ schemes into six classes for systematic comparison. In Section 6, for further comparison in fairness among the classes, we design a simulation to test the fairness properties of these classes. Finally, the conclusion is given in Section 7. The proofs of all theorems are listed in Appendix A.

2. Fair queueing

The goal of FQ is to allocate bandwidth fairly to the flows sharing a channel. In this section, we first introduce the system model, and then formally describe the goal of FQ and the definitions of fairness, bounded delay, and efficiency. Finally, we describe the theoretic FQ scheme, GPS, which helps understanding the design of practical FQ schemes.

2.1. System model

Assume at most N flows share the channel. We model the flows and the channel as a queueing system. In the system there is a non-preemptive, work-conserving server to send the packets from the flows. Each flow, with a separate FIFO packet queue, is associated with a reserved bandwidth. Each queue is assigned a dynamic priority; the smaller the priority values the higher the queue priority. The packet at the head of the highest-priority queue is first moved into the server, and then the queue priority is updated immediately.

In the following, we give some definitions and notations used.

Definition 1. A flow f is said to be *backlogged* during time interval $[t1,t2]$ (including $t1$ and $t2$) if the queue for f is never empty during $[t1,t2]$.

Definition 2. A system *busy period* is a maximal interval of time during which the server is never idle.

2.2. Goal of fair queueing

The goal of FQ is to share the server capacity among all backlogged flows in proportion to their reserved rates. Thus, if no flow changes status during $[t1,t2]$, the bandwidth $G.f(t1,t2)$ for flow f backlogged during $[t1,t2]$ is expected as

$$G.f(t_1,t_2) = C(R.f/\sum_{i\in B(t1,t2)} R.i). \tag{1}$$

With Eq. (1), FQ can reserve for f the bandwidth no less

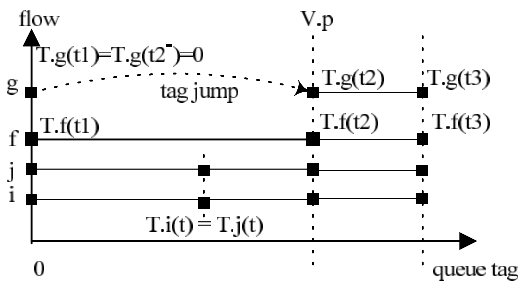


Fig. 1. Dynamics of queue tags in GPS.

than the reserved rate $R.f$, if

$$\sum_{i=0..N-1} R.i \le C. \tag{2}$$

With Eq. (1) and inequality (2), the criteria for FQ schemes are defined as follows.

Definition 3 (Fairness [8]). In a scheduling scheme, for any flow f backlogged during [$t1$,$t2$], its normalized service in the interval is defined as $W.f(t1,t2)/R.f$. The *fairness* between any two flows, i and j, backlogged during [$t1$,$t2$] is defined as the difference between the normalized services of the two flows. Let the fairness between the flows be denoted as $F.ij(t1,t2)$. Then, $F.ij(t1,t2) = |W.i(t1,t2)/R.i - W.j(t1,t2)/R.j.|$ The scheme is said to be fair (or to be with the fairness property), if there is an upper bound on $F.ij(t1,t2)$.

Definition 4 (Bounded delay [14,18]). Let $D.p$ be the delay of a packet p from a flow f through a shared server, which employs a scheduling scheme to serve a number of flows. On the other hand, let $d.p$ be the delay of the packet p of the same flow f through a solo server, which is dedicated for flow f with capacity $R.f$. The scheduling scheme employed by the shared server is with *bounded delay* if $D.p \le d.p + a$, where a is a small constant.

Definition 5 (Efficiency). [13]. The *efficiency* of a scheduling scheme is measured in two aspects: the complexities of *computing* and *scheduling*, which are the complexities for computing the priority of a queue and sorting the queue priorities, respectively.

To illustrate and compare FQ schemes, a basic scheme is shown as follows, named Basic_FQ, from which a lot of FQ schemes can be spawned.

(a) Consider a packet p of flow f arrives. Before the packet is appended to queue.f, the scheme checks if queue.f is empty. If queue.f is empty, priority $X.f$ and tag $T.f$ are updated as follows:

$$T.f \leftarrow \max\{V.p, X.f\}, \tag{3}$$

$$X.f \leftarrow T.f + L.p/R.f. \tag{4}$$

Here, $V.p$ denotes the virtual arrival time of packet p, whose definition varies from one scheme to another. On the contrary, if queue.f is not empty, then $X.f$ and $T.f$ are not updated.

(b) Once a packet departs from the server, the packet at the head of the highest priority queue is immediately moved into the server for transmission. Let queue.f be the highest priority queue. If queue.f remains non-empty, $X.f$ and $T.f$ are updated as follows:

$$T.f \leftarrow X.f, \tag{5}$$

$$X.f \leftarrow T.f + L.f/R.f. \tag{6}$$

In the next section, we introduce how to define $V.p$ to achieve the goal of FQ.

2.3. Theoretic scheme for fair queueing

GPS is the theoretic FQ scheme, which can achieve the goal of FQ. The scheme can be spawned from Basic_FQ by assuming that the queueing model is *fluid* and defining the virtual arrival time $V.p$ of a packet p as the queue tag of any flow backlogged on the arrival of p. GPS is not practical since it is based on the fluid model where the packet size is considered to be near zero. But the scheme contributes to the way for practical FQ schemes to define $V.p$.

Consider the Basic_FQ scheme. For each flow f, the tag $T.f$ is a step function of time with step size $L.f/R.f$ but possibly with a jump to the value of $V.p$, where packet p is from f. Whenever flow f becomes backlogged, assignment (3) is performed. If $X.f < V.p$, the new value of $T.f$ is from $V.p$, but not from $X.f$. In this case, we say that $T.f$ has a tag jump to $V.p$. Next, we show the dynamics of the flow tag with time in Fig. 1 to illustrate how GPS achieves the goal of FQ. Denote the value of $T.f$ at time t as $T.f(t)$. Let time $t1$ be the start of a busy period. At $t1$, each queue tag and queue priority is reset to zero. For any two flows, i and j, backlogged during [$t1$,$t2$], the queue tags of i and j are the same at any time t in [$t1$,$t2$], i.e.

$$T.i(t) = T.j(t). \tag{7}$$

This is because packet size is near zero and no tag jump occurs for i and j during [$t1$,$t2$].

Consider the bandwidth $G.f(t1,t2)$ while a flow f is backlogged during [$t1$,$t2$) and no flow changes the status during [$t1$,$t2$). Since $G.f(t1,t2)$ is $W.f(t1,t2)/(t2-t1)$, let us compute $W.f(t1,t2)$ first. Because $T.f$ has no tag jump during [$t1$,$t2$], we have $T.f \leftarrow T.f + L.f/R.f$ during [$t1$,$t2$]. Thus,

$$W.f(t1,t2) = (T.f(t2) - T.f(t1))R.f. \tag{8}$$

The sum of all $W.i(t1,t2)$, where $i \in B(t1,t2)$, is equal to the total data amount sent by the work-conserving server during [$t1$,$t2$], which is $(t2-t1)C$. For the flows i, $i \in B(t1,t2)$, their queue tags are zero at $t1$ and the same at $t2$ from Eq.

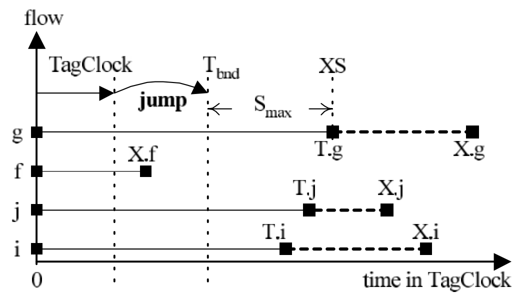


Fig. 2. Dynamics of TagClock in BTFQ.

(7). Since $f \in B(t1, t2)$, we have

$$(T.f(t2) - T.f(t1)) \sum_{i \in B(t1,t2)} R.i = (t2 - t1)C. \tag{9}$$

The allocated bandwidth $G.f(t1, t2)$ is thus equal to $C(R.f/\sum_{i \in B(t1,t2)} R.i)$. This matches the goal of FQ shown in Eq. (1).

Next, consider the bandwidth $G.g(t2, t3)$ when a flow g becomes backlogged at $t2$ and no flow changes status during ($t2$, $t3$]. At $t2$, a packet p from flow g arrives and causes g to become backlogged. The virtual arrival time *V.p* is thus equal to the queue tag of any flow backlogged at $t2^-$ (the time immediately before $t2$). According to assignment (3), *T.g* has a tag jump to *V.p* because *X.g* is 0 before $t2$. All flows backlogged during ($t2$, $t3$] still have the same queue tags at $t3$. With the result of Eqs. (8) and (9), the bandwidth $G.g(t2, t3)$ is equal to $C(R.g/\sum_{i \in B(t2,t3)} R.i)$. Therefore, the goal of FQ is achieved by GPS. In Fig. 1, $W.i(t1, t2)/R.i = W.j(t1, t2)/R.j$ from Eqs. (7) and (8). Thus, the fairness $Fij(t1, t2)$ is zero. Similarly, $Fij(t2, t3)$ is zero. So, GPS is an ideal FQ.

However, a feasible FQ scheme cannot be based on fluid model. In the practical packet model, the queue tags of backlogged flows may not be the same at any time due to packet size not near zero. Thus, it is difficult to determine an optimal value for defining *V.p*. The principle for achieving a better property in fairness is to define *V.p* as the value being as close as possible to the queue tags of all flows backlogged on the arrival of packet *p*. For example, the MSFQ scheme defines *V.p* as the smallest queue tag of all backlogged flows. In fact, the scheme has the properties of fairness,

```
**Enqueue** process: active on the arrival of a
    packet p from flow f
{
 If (p initializes a busy period) TagClock ← 0;
 If (queue.f is empty) {
   $T_{bnd}$ ← XS - $S_{max}$;
   TagClock ← max {TagClock, $T_{bnd}$ };
   T.f ← max {X.f, TagClock};
   X.f ← T.f + L.p/R.f;
   Insert (H, X.f);
 }
 Append (queue.f, p);
}
```

Fig. 3. Enqueue process of BTEQ.

```
**Dequeue** process: active on the departure of a
    packet
{
 if (no packet in the system) Reset (BTFQ);
 else {
   f ← FindTopPriorityFlow (H);
   p ← Head (queue.f);
   Transmit (p);
   XS ← X.f;
   Delete (H, X.f);
   If (queue.f is not empty) {
     T.f ← X.f;
     X.f ← T.f + L.f/ R.f;
     Insert (H, X.f);
   }
 }
}
```

Fig. 4. Dequeue process of BTFQ.

bounded delay and O(log *N*) complexity for computing the smallest queue tag, where *N* is the number of maximum flows allowed.

3. Bounded tag fair queueing scheme

We will describe the proposed scheme BTFQ, and show its properties of fairness, bounded delay and efficiency. BTFQ is spawn from Basic_FQ and defines the virtual arrival time *V.p* of packet *p* as the time of the special clock, named TagClock, when *p* arrives.

A snapshot of the dynamics of TagClock is shown in Fig. 2, where flow *f* is non-backlogged while flows *g*, *i*, and *j* are backlogged. Whenever a busy period starts, TagClock is reset to zero and then acts like a real clock. However, the time in TagClock may jump on the event that a packet *p* of a flow *f* arrives and causes the flow *f* to become backlogged. On the event, if the time in TagClock is less than the lower bound T_{bnd} of the queue tags of backlogged flows (*g*, *i*, and *j* in Fig. 2), the time in TagClock jumps to T_{bnd}. That is, $\text{TagClock} \leftarrow \max\{\text{TagClock}, T_{bnd}\}$. About the definition of *V.p*, if the arrival of packet *p* triggers a time jump in TagClock, *V.p* is defined as the time in TagClock immediately after the jump.

BTFQ consists of the Enqueue and Dequeue processes dealing with the arrival and departure of every packet, shown in Figs. 3 and 4. When a packet *p* from flow *f* arrives, the Enqueue process is active. If *f* has been backlogged, just append *p* to queue.*f*. On the contrary, if *f* has not been backlogged until *p* arrives, T_{bnd} is calculated as $XS - S_{\max}$ where *XS* is the priority of the packet currently served and $S_{\max}$ is $\max_{i=0\ldots N-1}\{L.i_{\max}/R.i\}$. If TagClock is smaller than T_{bnd}, it has a jump to T_{bnd}. Then, the virtual arrival time *V.p* is the time in TagClock. *T.f* and *X.f* are calculated and *X.f* is inserted into the ordered queue priority list *H*. Finally, packet *p* is append to queue.*f* to wait for service.

The Dequeue process is active on the departure of a packet. The function Reset(BTFQ) resets *XS* and all queue prioritys and tags to zero. FindTopPriorityFlow(*H*) returns the flow with the highest queue priority among the list H. Head(queue.f) removes the packet at the head of queue.f and then returns it. Transmit(p) starts the transmission of packet p. Finally, $T.f$ and $X.f$ are updated if queue.f is not empty.

As for the lower bound T_{bnd} of all backlogged queue tags, we explain it as follows.

Theorem 1. In BTFQ, for any currently backlogged flow f, its queue tag $T.f$ is no less than the priority XS of currently served packet minus $S_{\max}$. Thus T_{bnd} is assigned $XS - S_{\max}$.

Next, we show the properties of fairness, bounded delay, and efficiency as follows.

3.1. Bounded delay of BTFQ

To analyze the property of BTFQ in bounded delay, we show BTFQ belongs to the class of Guaranteed Rate (GR) scheduling schemes [9], all of which are with bounded delay. An algorithm to assign packets timestamps is introduced for describing the definition of the GR class as follows.

Let p and p' be the packets from flow f and p' arrive immediately preceding p. GRC(p) and A(p) are the timestamp and the arrival time of packet p, respectively. GRC(p) is computed as follows. (Here, GRC(p') is zero if p is the first packet of flow f.)

$$\mathrm{GRC}(p) \leftarrow \max\{\mathrm{A}(p), \mathrm{GRC}(p')\} + L.p/R.f.$$

With the above algorithm, the definition of the class of GR scheduling schemes is as follows.

Definition 6. A scheduling scheme belongs to the GR class if it guarantees that each packet p will depart from the server by GRC(p) + β, where β is a constant depending on the scheduling scheme and the server.

To show BTFQ belongs to GR class, we show the departure time of packet p is at most the value of the priority of p, $X.p$ plus L_{max}/C. Then, we show that $X.p$ is no greater than GRC(p). Thus the departure time of p is by GRC(p) + $L_{\max}/C$.

Theorem 2. If packet p of flow f departs from the BTFQ server (i.e. the server employing BTFQ) at time z in TagClock, z is at most the priority value of p, $X.p$ plus $L_{\max}/C$. That is,

$$z \le X.p + L_{\max}/C. \tag{10}$$

Theorem 3. BTFQ belongs to the GR class. That is, a packet p of flow f will depart from the BTFQ system by $GRC(p) + \beta$, where β is $L_{\max}/C$.

Corollary 1. BTFQ is with bounded delay.

Proof. From Theorem 3, BTFQ is a GR scheme. Since GR scheme is with bounded delay, so is BTFQ. Proven.

With the method from GR framework, we can easily determine the end-to-end delay bound for a flow f, if f is constrained by the Leaky Bucket control.

Theorem 4. Let flow f be constrained by the Leaky Bucket with parameter $(\sigma.f, R.f)$ and it passes through K BTFQ servers each of which has capacity C_i, $1 \le i \le k$. The end-to-end delay $D_{E2E}(f)$, if link propagation delays are not counted in, of the packet of f is bounded as

$$D_{E2E}(f) \le \sigma.f/R.f + (K-1)L.f_{\max}/R.f + \sum_{i=1\ldots K} L_{\max}/C_i. \tag{11}$$

The first term $\sigma.f/R.f$ is the delay that a packet p encounters the maximum burst size queued in queue.f of some server j along the path of f. The remaining terms result from the maximum delay, $L.f_{\max}/R.f + L_{\max}/C_i$, for the packet p through each server i where $i \neq j$.

3.2. Fairness of BTFQ

Recall the GPS scheme. The fairness $F.ij(t1, t2)$ is zero, for any two flows i and j backlogged during [$t1$,$t2$], since the queue tags of all backlogged flows are the same at any time. In BTFQ, all these tags can be bounded as shown in Theorem 5. With this theorem we introduce the upper bound of the fairness $F.ij(t1, t2)$ for BTFQ in Theorem 6.

Definition 7. Let current time be t in Tagclock. Define packet $e(t)$ as the latest one with the largest priority value among the packets entering the server from the start of current busy period until time t.

For example, current busy period starts at 0, current time is 10 in Tagclock, and five packets, $p1$, $p2$, $p3$, $p4$, and $p5$, have entered the server so far. The information for each packet, (packet priority, enter-server time), is as follows: $p1$(3, 0), $p2$(15, 2), $p3$(9, 3), $p4$(15, 5), $p5$(8, 8). $p2$ and $p4$ have the largest priority value 15, but $p4$ is the latest. Thus, e(10) is $p4$.

Theorem 5. For the BTFQ system in the current busy period, if current time is t in TagClock and $e(t)$ is packet r, then for each flow f backlogged at t the queue tag $T.f$ is bounded as

$$X.r - S_{\max} \le T.f \le X.r + L_{\max}/C. \tag{12}$$

Theorem 6. For the BTFQ scheme, the fairness

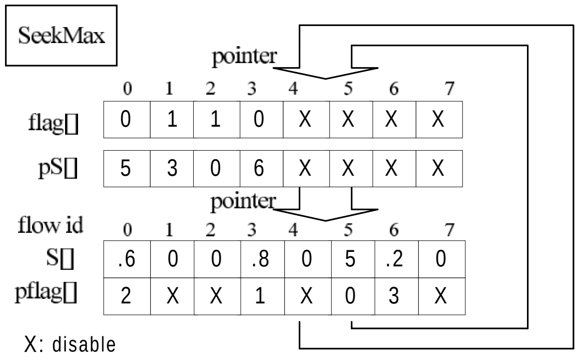


Fig. 5. SeekMax system.

$F.ij(t1,t2)$ between any two flows i and j backlogged during time interval $[t1,t2]$ is bounded as

$$F.ij(t1,t2) \le S_{\max} + \max\{L.i_{\max}/R.i, L.j_{\max}/R.j\} + L_{\max}/C. \tag{13}$$

3.3. Efficiency of BTFQ

From Definition 5, the efficiency of a scheduling scheme is evaluated on the complexities for computing and scheduling queue priority. In BTFQ for computing the queue priority $X.f$ of a flow f, the most complex operation in Enqueue and Dequeue is to compute $S_{\max}$. However, this operation is executed only when a flow is established or released. Thus, the complexity to compute $X.f$ is O(1) when a packet arrives or departs. On the other hand, to schedule $X.f$ into the ordered list H with at most N elements, the complexity is at least O(log N). Let H be a heap. The scheduling complexity is thus O(log N).

In fact, BTFQ also achieves O(loglog N) complexity with the method in the LFVC scheme [12], with which all the priorities of the currently backlogged queues are mapped into a finite integer range [0,M], where M is O(N). From the result of [9], a subset of [0,M] can be kept ordered by the insertion or deletion of each element in O(loglog M). However, the price paid is a more complex data structure than heap.

From Sections 3.1–3.3, BTFQ is rather attractive in fairness, bounded delay, and efficiency, especially for high speed network due to O(1) computing complexity. Further, BTFQ can be enhanced in the fairness bound by some modification.

4. BTFQ+

BTFQ + is proposed to improve the fairness property of BTFQ. Recall the BTFQ scheme, which defines the virtual arrival time $V.p$ of a packet p as the time in TagClock when p arrives. From the GPS scheme, the principle for achieving a better fairness property is to define $V.p$ as the value being as close as possible to the queue tags of all backlogged flows. Thus, BTFQ lets the time in TagClock jump to the lower bound T_{bnd} of all backlogged queue tags for being close to these tags. Evidently, it is the T_{bnd} that affects the fairness property. Therefore, the goal of BTFQ + is to find a new T_{bnd}, which is larger than the T_{bnd} in BTFQ. T_{bnd} in BTFQ is the value of the served packet priority $XS - S_{max}$. To enlarge the value of T_{bnd}, we replace $S_{\max}$ with $S_{\max+}$, where $S_{\max+}$ is the maximum of $L.i_{\max}/R.i$ of the flows currently backlogged and served. Clearly, $S_{\max}$ is no less than $S_{\max+}$ since $S_{\max}$ is the maximum of $L.i_{\max}/R.i$ of all flows i.

Thus, BTFQ + is modified from BTFQ by replacing only the assignment of T_{bnd} in the Enqueue process of BTFQ with the following:

$$T_{\text{bnd}} \leftarrow XS - S_{\max+}. \tag{14}$$

All the definitions, theorems and corollaries in BTFQ are still suitable for BTFQ+. Although the fairness is improved, the price paid is O(log N) time for computing $S_{\max+}$ since the backlogged flow set may change after a packet arrives or departs. This leads to an O(log N) computing complexity. To compensate the loss of efficiency, we design a supporting hardware to compute $S_{\max+}$. The computation time, if $N \le 4 \times 10^9$, is no longer than that of one ADD operation in a 32-bit adder.

4.1. Supporting Hardware for BTFQ+

The hardware, named SeekMax, is designed to compute $S_{\max+}$ when backlogged flow set changes. Assume at most N flows are in the system. Let $S[f]$ denote $L.f_{\max}/R.f$ for flow f. To find $S_{\max+}$ rapidly, the flows are sorted according to their $S[\ ]$. Let the sorted flow list be $(f_0, f_1,\ldots,f_{N-1})$ where $S[f_i] \ge S[f_j]$ if $0 \le i < j < N$. The state of flow f_i is indicated by the flag, flag[i]. If f_i is backlogged or served currently, flag[i] is 1; otherwise, 0. $S_{\max+}$ is the maximum of all the $S[f_i]$ of the flows f_i with flag[i] being 1 for $0 \le i < N$. To seek $S_{\max+}$ is to find the first flag[i] being 1, where i starts from 0. If such flag[i] is found, then $S_{\max+}$ is $S[f_i]$.

For example, in Fig. 5 there are flows 0, 3, 5, and 6 in the system. Here, $S[f]$ being 0 represents that flow f does not exist. The sorted flow list (f_0, f_1, f_2, f_3) is (5, 3, 0, 6), since $S[5] > S[3] > S[0] > S[6]$. Thus the flags of flows 5, 3, 0, and 6 are stored in flag[0] through flag[3]. In the example,

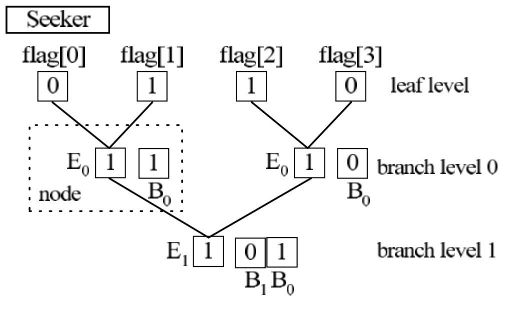


Fig. 6. Seeker of the SeekMax system.

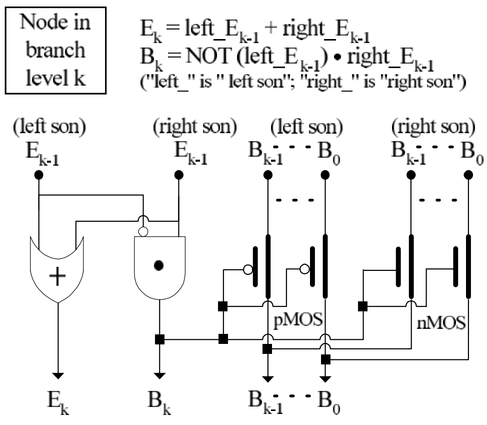

Node in branch level k
E_k = left_E_{k-1} + right_E_{k-1}
B_k = NOT (left_E_{k-1}) • right_E_{k-1}
("left_" is " left son"; "right_" is "right son")
(left son) E_{k-1}
(right son) E_{k-1}
(left son) B_{k-1} - - B_0
(right son) B_{k-1} - - B_0
+
•
pMOS
nMOS
E_k
B_k
B_{k-1} - - B_0

Fig. 7. Node logic in Seeker.

since the first flag being 1 is flag[1], $S_{\max+}$ is $S[f_1]$, i.e. $S[3]$. To keep the correspondence between $S[\ ]$ and flag[ ], two pointer arrays *p*flag[ ] and *pS*[ ] are for *S*[ ] and flag[ ], respectively. For example, flag[1] knows its correspondent, *S*[3], by the pointer in *pS*[1]. *S*[3] knows its correspondent, flag[1], by the pointer in *p*flag[3].

To rapidly find the first flag[*i*] being 1 where *i* starts from 0, Seeker with a binary-tree structure is created in SeekMax. A 4-flag Seeker is shown in Fig. 6. In Seeker, flags are in the leaf level. For the flags in a subtree, the first flag being one is indicated by the bits, E_k and $B_kB_{k-1}\ldots B_0$ in the root of the subtree, where the root is in branch level *k*. If E_k is 1, then there is a flag being one in the subtree. In this case, $B_kB_{k-1}\ldots B_0$ is the binary code of the index of the first flag being one in the subtree. For example, the B_1B_0 of the root in Fig. 6 being 01 shows the first flag being 1 is flag[$(01)_2$].

The values of E_k, B_k, and $B_{k-1}\ldots B_0$ of a node are computed from the E_{k-1} and $B_{k-1}\ldots B_0$ of the sons of the node. The computation logic is shown in Fig. 7. If one of the E_{k-1} of the sons is 1, E_k is 1. If the E_{k-1} of the left son is one, B_k is 0; otherwise, B_k is one if the E_{k-1} of the right son is 1. The $B_{k-1}\ldots B_0$ of the node is copied from that of the left or right son according to B_k being zero or one. Based on the VLSI design [7], this can be done by using *p*MOS and *n*MOS gates. In Fig. 7, if B_k is 0, *p*MOS will open; otherwise, close. The phase of *n*MOS is opposite to that of *p*MOS.

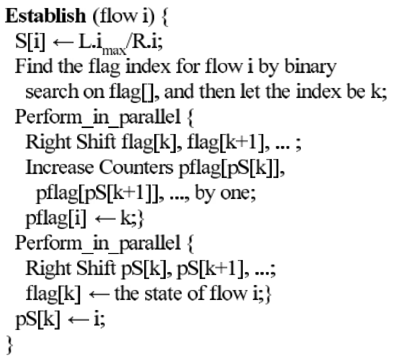
To compare the node of Seeker with a one-bit full adder, let bits A_{k-1} and B_{k-1} be the addends and bit C_{k-1} be the carry_in of the adder while S_k and C_k is the sum and carry_-out, respectively. The Boolean logic for the adder is:

$$S_k = C'_{k-1} \times (A'_{k-1} \times B_{k-1} + A_{k-1} \times B'_{k-1}) + C_{k-1} \times (A'_{k-1} \times B'_{k-1} + A_{k-1} \times B_{k-1}), \tag{15}$$

$$C_k = A_{k-1} \times B_{k-1} + C_{k-1} \times (A_{k-1} + B_{k-1}). \tag{16}$$

From Eq. (16), C_k is obtained after three operations, one ' × ' and two ' + '. Consider the node of Seeker in Fig. 7. $B_kB_{k-1}\ldots B_0$ is obtained after three operations, 'NOT', ' × ', and 'gate'. However, the delay of 'gate' is smaller than those of ' × ' and ' + ' in VLSI. Thus, the delay for a node is smaller than that for a one-bit full adder. Thus, Seeker on finding the first flag being one among *N* flags is faster than the $\lceil \log N \rceil$-bit adder on performing an ADD operation.

Finally, consider the time when a flow is established or released by the BTFQ + system. At this time, insertion or deletion is enforced on flag[ ]. For the consideration of speed, we implement flag[ ] as the shifter of *N* bits, *pS*[ ] as the shifter of *N* words, and *p*flag[ ] as the array of *N* counters. When a flow is established, SeekMax finds an *S*[*i*] being zero and then assigns *i* as the identifier of the flow. The processes of establishment and release of flow *i* in SeekMax are shown in Figs. 8 and 9, respectively.



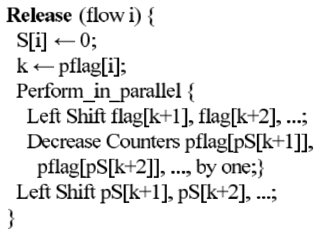
```
Establish (flow i) {
  S[i] ← L.i_max/R.i;
  Find the flag index for flow i by binary
    search on flag[], and then let the index be k;
  Perform_in_parallel {
    Right Shift flag[k], flag[k+1], ... ;
    Increase Counters pflag[pS[k]],
      pflag[pS[k+1]], ..., by one;
    pflag[i] ← k;}
  Perform_in_parallel {
    Right Shift pS[k], pS[k+1], ...;
    flag[k] ← the state of flow i;}
  pS[k] ← i;
}
```

Fig. 8. Flow establishment in SeekMax.

The establishment of a flow in SeekMax requires O(log *N*) time due to the binary search for a flag index for the flow. As to the instructions performed in parallel such as "Right Shift" and "Increase Counters", they need only one hardware clock cycle. On the contrary, the release of a flow requires O(1) time. These time complexities are required at the call level only. At the packet level, the time complexity of SeekMax is for finding $S_{\max+}$ by Seeker. A*N*-flag Seeker requires at most the time of one ADD operation in an $\lceil \log N \rceil$-bit adder.

5. Comparison

We compare BTFQ and BTFQ + with some well-known schemes in three criteria. The first is the bound of fairness $F.ij(t1,t2)$. For example, from Theorem 6, $F.ij(t1,t2)$ of



```
Release (flow i) {
  S[i] ← 0;
  k ← pflag[i];
  Perform_in_parallel {
    Left Shift flag[k+1], flag[k+2], ...;
    Decrease Counters pflag[pS[k+1]],
      pflag[pS[k+2]], ..., by one;}
  Left Shift pS[k+1], pS[k+2], ...;
}
```

Fig. 9. Flow release in SeekMax.

BTFQ is bounded by $\max_{k=0\ldots N-1}\{L.k_{\max}/R.k\} + \max\{L.i_{\max}/R.i, L.j_{\max}/R.j\} + L_{\max}/C$.

The second is delay bound. Consider a flow *f* constrained by the Leaky Bucket $(\sigma.f, R.f)$ and passing through *K* servers each of which has the capacity *C* and employs the same scheme with bounded delay. With the GR framework [9], flow *f* has an end-to-end delay bound, $\sigma.f/R.f + K\tau$, where τ is named the latency at a server. From Theorem 4, for BTFQ, the end-to-end delay of flow *f* is less than $\sigma.f/R.f + K(L.f_{\max}/R.f + L_{\max}/C)$. Hence for BTFQ τ is $L.f_{\max}/R.f + L_{\max}/C$. We let latency τ be the criterion for delay bound.

The third is efficiency. We compare the computing and scheduling complexities among these schemes. We summarize the performance data for these schemes in Table 1 [15]. Noticeably, all the schemes have the scheduling complexity of O(log *N*), except LFVC.

5.1. Classification and comparison

The schemes in Table 1 are classified into six classes according to their fairness bounds. In Class 0, GPS is an ideal FQ scheme with fairness bound being zero. Since it is based on the fluid model, the latency τ is zero. GPS is good but not practical. Thus we do not discuss its complexities. In Class 1, PGPS schedules packets in the practical packet model by simulating GPS in real time. Thus it is the scheme with very good property of fairness though it has larger fairness bound than some schemes. Its computing complexity is O(*N*); therefore, it is not suitable for high speed networks even though it provides a small latency.

For the Class 2 schemes, they have the good properties of fairness since the bounds of $F.ij(t1, t2)$ depend on flow *i* and *j* and/or the constant $L_{\max}/C$ only. SCFQ and SFQ have the smallest computing complexity, O(1). However, the latencies they provide increase with the number of flows. LFVC has not this problem, but it needs a more complex data structure.

The fairness properties of the Class 3 schemes are worse than those of the Class 2, since $F.ij(t1, t2)$ of the Class 3 not only depends on flow *i* and *j* but also on the maximum of the $L.k/R.k$ or $L.k_{\max}/R.k$ of some flows *k*. The latencies that the Class 3 schemes provide are as good as those the PGPS provide while their computing complexities are O(log *N*). Thus BTFQ + is designed to combine with a low-cost, high-speed hardware for reducing the computation time.

The Class 4 schemes have larger fairness bounds than the schemes of other classes except Class 5. They provide the same latency as PGPS and have the computing complexity of O(1). In the class, BTFQ has a smaller fairness bound than FFQ. As for Class 5, VC is not fair, since it has no fairness bound.

BTFQ suits the real-time applications in high speed networks due to the good properties in latency and efficiency. Although BTFQ is a little bit weak in fairness, BTFQ + improves the fairness of BTFQ and combines with the hardware SeekMax to reduce the computation time. As a whole, BTFQ + is good in fairness, bounded delay and efficiency at the same time. Notably, SPFQ has the same fairness bound as BTFQ, but it is classified into Class 3. The reason is explained in Section 5.1. In addition, from Table 1, the performance of LFVC is good as a whole. LFVC is compared further with the proposed schemes in Section 5.2.

5.2. Comparison with the SPFQ scheme

We describe SPFQ by means of the scheme Basic_FQ, mentioned in Section 2.2. SPFQ maintains a system clock, named system potential (SP), to keep the values of backlogged queue tags close. When a packet *p* arrives, its virtual arrival time *V.p* is defined as the time in SP. SP (in SPFQ) and TagClock (in BTFQ) are alike; however, they are different in the values that their times jump to and in the events that trigger the occurrences of the time jumps. The time in SP may jump on the event that a packet departs from the server. On the event, if the time in SP is smaller than the smallest queue tag among the backlogged queues, the time jumps to the tag value. That is, $\text{SP} \leftarrow \max\{\text{SP}, \min_{i:\text{backlogged flow}}\{T.i\}\}$. For finding out the smallest queue tag in O(1) time, SPFQ needs to maintain the backlogged queue tags as an ordered list such as a heap. Thus it takes O(log *N*) time to insert or delete a queue tag in the ordered list. When a packet *p* from a flow *f* arrives and causes flow *f* to become backlogged, the queue tag *T.f* and queue priority *X.f* are updated immediately according to assignments (3) and (4). Next, *T.f* and *X.f* are inserted into the ordered queue tag and queue priority lists, respectively. These operations need O(log *N*) time. Clearly, both the computing complexity and scheduling complexity are O(log *N*).

SPFQ bounds the fairness *Fij*(*t*1, *t*2) under $\max_{k=0\ldots N-1}\{L.k_{\max}/R.k\} + \max\{L.i_{\max}/R.i, L.j_{\max}/R.j\} + L_{\max}/C$. However, we believe the fairness of SPFQ has a tighter bound, $\max_{k\in B(t)}\{L.k/R.k\} + \max\{L.i_{\max}/R.i, L.j_{\max}/R.j\} + L_{\max}/C$, where $B(t)$ is the set of the flows backlogged at time *t*. Our belief arises from the way in which SPFQ defines the virtual arrival time *V.p* of a packet *p*. According to the basic scheme Basic_FQ, the way to define *V.p* dominates the fairness property of an FQ scheme. We find that the way of SPFQ to define *V.p* is similar to those of TSS and MSFQ. TSS has a ShiftClock like TagClock (in BTFQ). When a packet *p* from a flow arrives and causes the flow to become backlogged, ShiftClock is updated as follows: $\text{ShiftClock} \leftarrow \max\{\text{ShiftClock}, \min_{i:\text{backlogged flow}}\{T.i\}\}$. Then *V.p* is defined as the time in ShiftClock. MSFQ has no system clock to help define *V.p*. Thus it defines *V.p* as $\min_{i:\text{backlogged or served flow}}\{T.i\}$ when *p* arrives. Obviously, the fairness bounds of SPFQ, TSS, and MSFQ only depend on the queue tags of the flows currently backlogged and/or served. Therefore the first term $\max_{k=0..N-1}\{L.k_{\max}/R.k\}$ of the fairness bound declared by SPFQ should be modified to

Table 1
Summary of the performances of scheduling schemes (the symbol # in the top of the first column represents the class number. $A(t)$ is the set of the flows served and backlogged at a time t)

#	Scheme	Fairness bound for $F.ij(t1, t2)$	Latency τ	Computing	Scheduling
0	GPS	0	0	–	–
1	PGPS	$\max\left\{\frac{L.i_{\max}}{R.i} + U.i, \frac{L.j_{\max}}{R.j} + U.j, \frac{L.i_{\max}}{R.i} + \frac{L_{\max}}{R.j}, \frac{L.j_{\max}}{R.j} + \frac{L_{\max}}{R.i}\right\}$, where $U.i = \min\left\{(N-1)\frac{L_{\max}}{R.i}, \max_{k=0}^{N-1}\left\{\frac{L.k_{\max}}{R.k}\right\}\right\}$	$\frac{L.i_{\max}}{R.i} + \frac{L_{\max}}{C}$	O(N)	O(log N)
2	SCFQ	$\frac{L.i_{\max}}{R.i} + \frac{L.j_{\max}}{R.j}$	$\frac{L.i_{\max}}{R.i} + \sum_{k=0,k\neq i}^{N-1} \frac{L.k_{\max}}{C}$	O(1)	O(log N)
	SFQ	$\frac{L.i_{\max}}{R.i} + \frac{L.j_{\max}}{R.j}$	$\sum_{k=0}^{N-1} \frac{L.k_{\max}}{C}$	O(1)	O(log N)
	LFVC	$2\left(\frac{L.i_{\max}}{R.i} + \frac{L.j_{\max}}{R.j} + \frac{L_{\max}}{C}\right)$	$\frac{L.i_{\max}}{R.i} + \frac{L_{\max}}{C}$	O(log log N)	Amortized O(log log N)
3	MSFQ	$\max\left\{\frac{L.i_{\max}}{R.i} + \frac{L.j_{\max}}{R.j}, \max_{k\in A(t)}\left\{\frac{L.k}{R.k}\right\} + \max\left\{\frac{L.i_{\max}}{R.i}, \frac{L.j_{\max}}{R.j}\right\}\right\}$	$\frac{L.i_{\max}}{R.i} + \frac{L_{\max}}{C}$	O(log N)	O(log N)
	TSS	$\max_{k\in B(t)}\left\{\frac{L.k}{R.k}\right\} + \max\left\{\frac{L.i_{\max}}{R.i}, \frac{L.j_{\max}}{R.j}\right\} + \frac{L_{\max}}{C}$	$\frac{L.i_{\max}}{R.i} + \frac{L_{\max}}{C}$	O(log N)	O(log N)
	SPFQ	$\max_{k=0}^{N-1}\left\{\frac{L.k_{\max}}{R.k}\right\} + \max\left\{\frac{L.i_{\max}}{R.i}, \frac{L.j_{\max}}{R.j}\right\} + \frac{L_{\max}}{C}$	$\frac{L.i_{\max}}{R.i} + \frac{L_{\max}}{C}$	O(log N)	O(log N)
	BTFQ+	$\max_{k\in A(t)}\left\{\frac{L.k_{\max}}{R.k}\right\} + \max\left\{\frac{L.i_{\max}}{R.i}, \frac{L.j_{\max}}{R.j}\right\} + \frac{L_{\max}}{C}$;	$\frac{L.i_{\max}}{R.i} + \frac{L_{\max}}{C}$	O(log N)	O(log N)
4	FFQ	$\frac{2F}{C} + \max\left\{\frac{L.i_{\max}}{R.i}, \frac{L.j_{\max}}{R.j}\right\}$, where $\frac{F}{C} = \max_{k=0}^{N-1}\left\{\frac{L.k_{\max}}{R.k}\right\}$	$\frac{L.i_{\max}}{R.i} + \frac{L_{\max}}{C}$	O(1)	O(log N)
	BTFQ	$\max_{k=0}^{N-1}\left\{\frac{L.k_{\max}}{R.k}\right\} + \max\left\{\frac{L.i_{\max}}{R.i}, \frac{L.j_{\max}}{R.j}\right\} + \frac{L_{\max}}{C}$	$\frac{L.i_{\max}}{R.i} + \frac{L_{\max}}{C}$	O(1)	O(log N)
5	VC	∞	$\frac{L.i_{\max}}{R.i} + \frac{L_{\max}}{C}$	O(1)	O(log N)

$\max_{k\in B(t)}\{L.k/R.k\}$. SPFQ is thus classified into Class 3. From Table 1, SPFQ, TSS, and MSFQ have better fairness than BTFQ + . Nevertheless, by sacrificing a little fairness, BTFQ + is designed to combine with a simple hardware to reduce the computation time.

5.3. Comparison with the LFVC scheme

First, we describe the LFVC scheme by the Basic_FQ scheme. LFVC maintains a system variable, named ServerTime, to keep the values of backlogged queue tags close. When a packet p arrives, its virtual arrival time $V.p$ is defined as the value of ServerTime. ServerTime is updated in the following way. The backlogged flows are divided into two areas, high priority area (HA) and low priority area (LA) according to their queue tags. If the queue tag $T.f$ of the backlogged flow f is less than the value of ServerTime plus the constant $L_{\max}/C$, flow f is put into HA; otherwise, LA. In HA and LA, the flows are sorted according to their queue priorities and tags, respectively. The flow with the smallest queue priority value in HA is served first. If the queue tag of the flow becomes no less than ServerTime plus $L_{\max}/C$, the flow is moved into LA. Once there is no flow in HA and the value of ServerTime is less than the smallest queue tag in LA, denoted by $T_{\min}$, ServerTime will be set to $T_{\min}$. When the server starts transmitting a packet p, ServerTime is increased by $L.p/C$. The flows in LA with the queue tags less than ServerTime plus $L_{\max}/C$ are moved into HA immediately.

Secondly, we analyze the efficiency of LFVC. Because LFVC maps the queue priority values into an integer range [0,M], where M is O(N), it takes O(loglog N) time to insert or delete a flow in HA by the results of [9]. In the worst case, however, N flows need to be moved into HA from LA during one packet transmission. Thus the scheduling complexity is O(N loglog N). In fact, in Table 1, O(loglog N) is the amortized scheduling complexity (i.e. the average scheduling complexity on each packet transmission) because each packet is moved into HA at most once. BTFQ and BTFQ + can also map the queue priority values into an integer range [0,O(N)] and thus achieve O(loglog N) scheduling complexity. Their efficiencies are better than LFVC; however, their fairness properties are weaker than that of LFVC, since LFVC is a Class 2 scheme. To further understand the difference of fairness among the classes in Table 1, we design a simulation to evaluate the fairness degrees of these classes.

6. Simulation and results

It is necessary to create a way to evaluate the average degree of the fairness property of an FQ scheme. In this section, the fairness degree is defined first. Then, a simulation is designed to measure the fairness degrees of the schemes in the classes shown in Table 1 except Class 0 and 5.

6.1. Fairness degree

The method to capture the fairness degree is to measure the difference between ideal and practical scheduling of a scheme. Recall the Basic_FQ scheme in Section 2.2. When a flow f becomes backlogged, from assignment (3) the queue tag $T.f$ is assigned the maximum of the queue priority $X.f$ not yet updated and the $V.p$ where p is the packet whose arrival causes f backlogged. Consider the case that the value of $T.f$ is from $V.p$. Since different schemes define $V.p$ with different values, the values of $T.f$ in different schemes are different in this case. However, there should be an ideal scheduling value for $T.f$ to achieve the optimal fairness. Since the value of $T.f$ is from $V.p$, according to the principle of defining $V.p$ (Section 2.3) the ideal value of $T.f$ should much near the average of the queue tags of backlogged and served flows weighted by their reserved rates. Thus, the difference between the practical value and the ideal value of $T.f$ is defined as fairness difference.

Definition 8. For an FQ scheme spawn from the Basic_FQ scheme if at time t a packet p of flow f arrives such that f becomes backlogged and the new value of the queue tag $T.f$ is from $V.p$ (referred to assignment (3)), then the *fairness difference* FD is defined as follows:

$$\mathrm{FD} = \Big|T.f - [1/2(T.q + X.q) * R.g/\Omega + \sum_{k\in B(t),k\neq g} T.k * R.k/\Omega]\Big|, \tag{17}$$

where q from flow g is the packet being served when p arrives and

$$\Omega = R.g + \sum_{k\in B(t),k\neq g} R.k.$$

The fairness degree of an FQ scheme is thus defined as in the following definition.

Definition 9. For an FQ scheme, let the fairness difference FD be the random variable given as Definition 8. Let E[FD] be the expected value of FD and $S_{\max}$ be $\max_{i=0\ldots N-1}\{L.i_{\max}/R.i\}$. Then, the *fairness degree* of the scheme is defined as:

$$\text{Fairness degree} = 1 - E[\mathrm{FD}]/S_{\max}. \tag{18}$$

It is reasonable to normalize E[FD] by $S_{\max}$ since a good FQ scheme should keep the difference of the queue tags of any two backlogged flows no larger than $S_{\max}$ plus a small constant.

For the GPS scheme, E[FD] is zero since all the queue tags of backlogged flows are the same at any time. Thus its fairness degree is 100%. To understand the fairness degree of each class shown in Table 1, we perform simulation on the scheme with the smallest fairness bound in each class. For Class 2, we select SCFQ while SFQ has the same fairness bound. For Class 3, we select TSS while MSFQ has almost the same fairness bound. For Class 4, we select BTFQ. In addition, BTFQ + is selected for exhibiting the improvement on the fairness of BTFQ.

We briefly introduce the selected schemes as follows. Recall the Basic_FQ scheme. From assignment (3), various schemes are spawn by defining $V.p$ with different values, where p is a packet from a flow f whose arrival causes flow f backlogged. For PGPS, which simulates a GPS scheduler, $V.p$ is the virtual arrival time of packet p in the simulated GPS scheduler. For SCFQ, $V.p$ is the priority value of the packet being served in the server. For TSS, there is a system clock, ShiftClock, like TagClock of BTFQ. When p arrives, ShiftClock is set to max{ShiftClock, the smallest backlogged queue tag}. Then $V.p$ is the time in ShiftClock. For BTFQ and BTFQ + , when p arrives, TagClock is set to max{TagClock, T_{bnd}}, where T_{bnd} is the lower bound of the backlogged queue tags. Then $V.p$ is the time in TagClock. BTFQ and BTFQ + are different in their T_{bnd}. The T_{bnd} of BTFQ is no greater than that of BTFQ + . From GPS, we know that the better the fairness of a scheme, the closer the $V.p$ of the scheme to the backlogged queue tags. For TSS, BTFQ, and BTFQ + , since ShiftClock acts like TagClock,

Table 2
Traffic characteristics of input flows for Experiment 1 (the unit of avg. rate, const. rate, and peak rate is in Mbps. The unit of avg. burst and bucket size is packet and token, respectively)

Flow	Expe_1	0	1	2	3	4	5	6	7	8	9
0 (Poiss)	Avg. rate	0.064	0.064	0.064	0.064	0.064	0.064	0.064	1.5	3	5
1 (Const)	Const. rate	0.064	0.064	0.064	0.064	0.064	0.064	0.064	1.5	3	5
2 (IDP)	Avg. rate	0.064	0.064	0.064	0.064	0.064	0.064	0.064	1.5	3	5
	Peak rate	0.64	0.64	0.32	0.32	0.192	0.192	0.128	7.5	9	10
	Avg. burst	10	20	40	25	5	10	20	40	5	25
3 (PLB) (token size	Avg. rate	0.064	0.064	0.064	0.064	0.064	0.064	0.064	1.5	3	5
	Peak rate	0.64	0.64	0.32	0.32	0.192	0.192	0.128	7.5	9	10
53 bytes)	Bucket size	10	20	40	25	5	10	20	40	5	25

Table 3
Traffic characteristics of input flows for Experiment 2

Flow	Expe_2	0	1	2	3	4	5	6	7	8	9
0 (Poiss)	Avg. rate	1	1	1	1	1	1	1.5	2	5	10
1 (Const)	Const. rate	1	1	1	1	1	1	1.5	2	5	10
2 (IDP)	Avg. rate	1	1	1	1	1	1	1.5	2	5	10
	Peak rate	10	10	5	5	2	2	7.5	9	10	20
	Avg. burst	10	20	40	25	5	10	20	40	5	25
3 (PLB) (token size 53 bytes)	Avg. rate	1	1	1	1	1	1	1.5	2	5	10
	Peak rate	10	10	5	5	2	2	7.5	9	10	20
	Bucket size	10	20	40	25	5	10	20	40	5	25

we have the following conclusions. The fairness degree of TSS is better than those of BTFQ and BTFQ + , since the smallest backlogged queue tag is no less than T_{bnd}. The fairness degree of BTFQ + is better than that of BTFQ, since the T_{bnd} of BTFQ + is no less than that of BTFQ.

6.2. Experiments

We perform simulation on the schemes, PGPS, SCFQ, TSS, BTFQ, and BTFQ + , and design three experiments, Expe_1, Expe_2, and Expe_3, to find out their fairness degrees. The key point of selecting the flow traffics is to sufficiently exhibit the difference among the fairness degrees of these schemes. From Definitions 8 and 9, the fairness degree of a scheme depends on the virtual arrival times $V.p$ of the packets p whose arrivals cause flows to become backlogged. The more the amount of such packets p, the more accurate the results of the experiments. Because the arrival of such a packet p causes the flow the packet comes from to become backlogged (from non-backlogged), a flow with a lot of arrivals of such packets p must alternate between backlog and non-backlog frequently. A flow with strong burstiness, which can be modeled by the heavy-tailed distributions [22,23], is not suitable for the experiments since a long burst would make the flow stay in backlog situation for very long time and reduce the alternating frequency between backlog and non-backlog. In the realistic world, a lot of flows are of strong burstiness and a great number of strong-burstiness flows must be created in order to generate sufficient packets p causing flows to become backlogged for exhibiting the full functioning of FQ schemes in the experiments. In fact, experiments with a large number of flows increase the difficulty of simulation. A tradeoff is to apply the Poisson, constant-rate, and short-burst flows to the experiments. These flows do not have long burst and thus can alternate between backlog and non-backlog frequently.

In each experiment we choose four types of traffic flows for testing: Poisson (Type 0), Constant Rate (Type

Table 4
Traffic characteristics of input flows for Experiment 3

Flow	Experiment 3	0	1	2	3	4	5	6	7	8	9
0 (Poiss)	Avg. rate	0.064	1	1	1	1	1	1.5	3	5	10
1 (Const)	Const. Rate	0.064	1	1	1	1	1	1.5	3	5	10
2 (IDP)	Avg. rate	0.064	1	1	1	1	1	1.5	3	5	10
	Peak rate	0.64	10	5	5	2	2	7.5	9	10	20
	Avg. burst	10	20	40	25	5	10	20	40	5	25
3 (PLB) (token size 53 bytes)	Avg. rate	0.064	1	1	1	1	1	1.5	3	5	10
	Peak rate	0.64	10	5	5	2	2	7.5	9	10	20
	Bucket size	10	20	40	25	5	10	20	40	5	25

Table 5
Statistics of fairness difference for Experiment 3 (unit of server rate: Mbps, $S_{max} = 6625$)

Expe_3	Server rate Load	115 0.854	110 0.893	105 0.936	100 0.983
WFQ	Avg.	30.4	31.0	33.2	31.4
	Stdev.	86.9	90.3	82.1	69.9
	Max.	6012.0	5491.9	6134.1	6301.5
SCFQ	Avg.	58.0	62.9	71.1	82.7
	Stdev.	140.7	139.3	133.1	97.7
	Max.	4968.8	5520.8	5506.7	5162.8
TSS	Avg.	50.2	70.8	142.0	480.7
	Stdev.	194.6	246.1	400.5	681.6
	Max.	5832.3	6071.4	6255.1	6249.9
BTFQ +	Avg.	54.0	84.3	195.2	833.9
	Stdev.	218.8	324.3	601.8	1265.9
	Max.	6449.7	6466.4	6482.6	6543.4
BTFQ	Avg.	87.35	155.3	384.5	1273.0
	Stdev.	367.7	563.0	970.7	1530.6
	Max.	6489.3	6492.4	6545.7	6567.2

1), Interrupt Deterministic Process (IDP) (Type 2), and Poisson with Leaky Bucket (PLB) (Type 3). For each type there are ten flows. Packets in the flows are of the same size, 53 bytes, which is the packet size of the ATM network. The traffic parameters of all flows are shown in Table 2–4.

IDP is the 2-state Markov Modulated Deterministic Process [10], depicting the traffic with 2 alternated states, "on" and "off". In the "on" state traffic is constant rate while in the "off" state it is zero rate. For example, for Flow[2,9] in Expe_1, its "on" state is long for 25 packets in average and introduces a peak rate of 10 Mbps. The flow in PLB is formed by a Poisson source constrained by a Leaky Bucket. For example, for Flow[3,9] in Expe_1, its Poisson source with average rate 5 Mbps is constrained by the Leaky Bucket whose token size is 53 bytes, token rate 5 Mbps, bucket size 25 tokens, and output rate 10 Mbps. In the three experiments, the greatest average burst among the IDP flows is 40 packets. These IDP flows are designed as the short-burst traffics. As for the PLB flows, they are the traffics of Poisson shaped to short burst and are designed for increasing the variation in testing traffic.

In Expe_1, the average rates of the flows are from 0.064 to 5 Mbps while those are from 1 to 10 Mbps in Expe_2. This represents that the testing is not only for the flows distributing on the narrow band but also for those on the broadband. Expe_1 and Expe_2 are designed to test the fairness degrees of the FQ classes. In Expe_3, most of the average rates of the flows are high (from 1 to 10 Mbps); however, a few are low (0.064 Mbps). This is designed to test the BTFQ + for exhibiting the improvement on the fairness of BTFQ.

For each scheme, the reserved rate of a flow is set to the average rate of the flow. To make various traffic loads to test a scheme, we change the server capacity, instead of adjusting the input flow rates. The results of Expe_1–Expe_3 are shown in Figs.10–12, respectively. PGPS has the highest fairness degree, about 98.5% in Fig. 10 and 93% in Fig. 11, tested by any traffic load. This represents that PGPS is the best scheme in fairness. For SCFQ, the fairness degree decreases with increasing traffic load. TSS, BTFQ, and BTFQ + have the same situation, too. However, their fairness degrees fall fast when load is over 95%. The reason is as follows.

Let *p* be the packet from flow *f* whose arrival causes flow *f* backlogged. From the definition of FD, the value of queue tag *T.f* is from *V.p*. Thus FD is the difference of the practical and ideal *V.p*. The practical *V.p* is the *V.p* defined by a FQ scheme. For example, SCFQ defines *V.p* as the priority value of the packet in server while TSS defines *V.p* as the time in ShiftClock. On the contrary, the ideal *V.p* is about the average of all backlogged queue tags weighted by the individual reserved rates from Definition 8.

On the high traffic load, a large number of flows with small reserved rate are backlogged, so the values of the backlogged queue tags distribute widely. Thus, the ideal

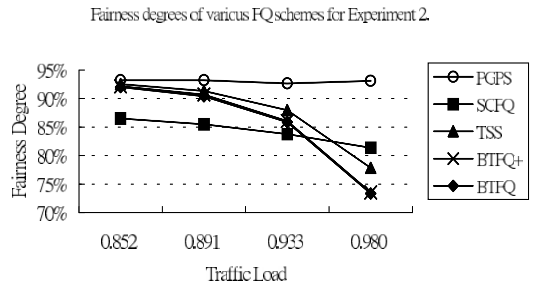


Fig. 11. Fairness degree in Experiment 2.

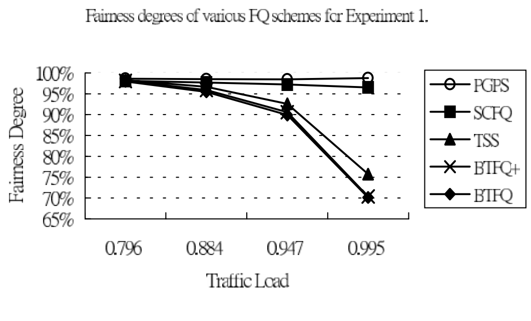


Fig. 10. Fairness degree in Experiment 1.

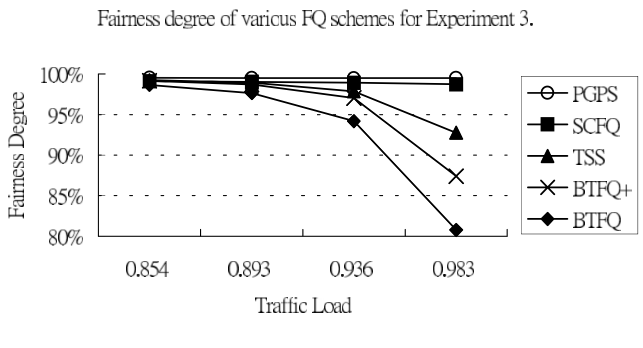


Fig. 12. Fairness degree in Experiment 3.

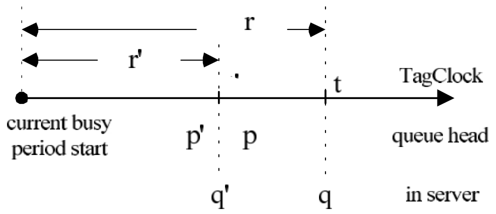


Fig. 13. Relation of packets and TagClock.

V.p is near the center of the backlogged queue tags. The *V.p* defined by TSS is farther to the ideal *V.p* than the *V.p* defined by SCFQ, since the *V.p* defined by TSS is the smallest backlogged queue tag. Thus the FD of TSS is much larger than that of SCFQ. This leads to that the fairness degree of TSS falls much faster than that of SCFQ on high traffic load. From the explanation in Section 6.1, the fairness degrees of BTFQ and BTFQ + are not as good as that of TSS. Thus the degrees of BTFQ and BTFQ + fall much faster on high traffic load, too. Nevertheless, the fairness degrees of the FQ schemes in Class 3 and 4 are not far from those in Class 1 and 2, if the traffic load is under 95%.

In Expe_1 and Expe_2, the fairness degree of BTFQ + is almost the same as that of BTFQ. The reason is as follows. In high traffic load there is often a flow with the smallest reserved rate (0.064 Mbps in Expe_1 and 1 Mbps in Expe_2) backlogged or served. Thus $S_{\max+}$ (the maximum of $L.f_{\max}/R.f$ of the served and backlogged flows *f*) is often equal to $S_{\max}$ (the maximum of $L.f_{\max}/R.f$ of all flows *f*). This results in that the T_{bnd} of BTFQ + is often equal to that of BTFQ. Therefore, the fairness degree of BTFQ + is often equal to that of BTFQ. However, in Expe_3, since the number of the flows with the smallest reserved rate (0.064 Mbps) is few, the probability that at least one such flow is backlogged or served is small even in high traffic load. Thus $S_{\max+}$ is often equal to 424 (= 424/1) while $S_{\max}$ is 6625 (= 424/0.064). The T_{bnd} of BTFQ + is hence often much greater than that of BTFQ. In this case, BTFQ + improves the fairness of BTFQ very much. This argument is verified by the simulation result listed in Table 5. For example, the averages of fairness difference of TSS, BTFQ + , and BTFQ are 70.8, 84.3, and 155.3 on the traffic load 0.893. Obviously, the fairness degree of BTFQ + is very close to that of TSS and far from that of BTFQ.

From the simulation, in general, BTFQ possess good fairness property under 95% traffic load. BTFQ + achieves much better fairness property than BTFQ when the combination of the input flows, such as that in Expe_3, leads to that $S_{\max+}$ is often much smaller than $S_{\max}$.

7. Conclusions

FQ is a rather attractive scheduling mechanism, which can fairly allocate bandwidth to the flows sharing a network channel. By controlling the bandwidth reserved for the flows and limiting their traffic characteristics, FQ can further guarantee bounded delay for the flows. In this paper, we propose a scheme BTFQ and its enhancement BTFQ + to implement the FQ mechanism. BTFQ is both fair and delay bounded. It is very efficient since it computes and schedules the priority of flow queue in the time complexities, O(1) and O(log *N*). BTFQ + is designed to improve the fairness property of BTFQ and combine with a low-cost, high-speed hardware for reducing the computation time. The hardware can complete the work trusted by BTFQ + in a time, which is shorter than the time of one ADD operation in a $\lceil \log N \rceil$-bit adder.

For comparison, we classify the proposed schemes and some well-known schemes into six classes according to their fairness bounds. Based on the criteria of fairness, bounded delay, and efficiency, we find no class is perfect in all aspects. Class 0 is not practical. Class 1 is not efficient. Class 2 has larger bounded delay or more complex structure. Class 3 has a little loss in efficiency and fairness. Class 4 has more loss in fairness. Class 5 is unfair.

BTFQ + and BTFQ are the schemes in Class 3 and Class 4, respectively. From Table 1, they provide the smallest bounded delay. In addition BTFQ and BTFQ + with the supporting hardware achieve the best efficiency. From the simulation results, BTFQ and BTFQ + have the fairness degrees near those of the schemes in Class 1 and 2, if traffic load does not exceed 95%. Therefore, BTFQ and BTFQ + have excellent performance. Since no practical FQ scheme is perfect till now, BTFQ and BTFQ + provide another choices.

Acknowledgements

The authors are very grateful to the anonymous reviewers for the comments and suggestions that helped improve the quality of this paper.

Appendix A

Theorem 1. In BTFQ, for any currently backlogged flow *f*, its queue tag *T.f* is no less than the priority *XS* of currently served packet minus $S_{\max}$. Thus T_{bnd} is assigned $XS - S_{\max}$.

Proof. The proof is shown by considering the relation of XS and the queue priority *X.f* of *f*.

Case 1. Consider $XS \le X.f$. Since $S_{\max} = \max_{i=0\ldots N-1}\{L.i_{\max}/R.i\}$, we have $(XS - S_{\max}) \le (X.f - L.f/R.f)$. Therefore, $XS - S_{\max} \le T.f$.

Case 2. Consider $XS > X.f$. The event that f becomes backlogged must occur during the transmission of the currently served packet. From the Enqueue process shown in Fig. 3, $T.f$ is assigned $\max\{X.f,\ \max\{TagClock, T_{\text{bnd}}\}\}$, where T_{bnd} has been assigned $XS - S_{max}$. Thus $T_{\text{bnd}} \le T.f$. That is, $XS - S_{\max} \le T.f$.

Theorem 2. If packet p of flow f departs from the BTFQ server (i.e. the server employing BTFQ) at time z in TagClock, z is at most the priority value of p, $X.p$ plus $L_{\max}/C$. That is,

$$z \le X.p + L_{\max}/C. \tag{A1}$$

Proof. In this theorem time is referred to TagClock. We first define a special time s, and then calculate how long after s packet p will depart. Let u be the time packet p becomes the head of queue.f. Time s is defined as the latest time, no later than u (i.e. $s \le u$), such that either of the following events occurs:

Event1. A busy period starts.
Event2. The time in TagClock jumps to T_{bnd}. (Thus $s = T_{\text{bnd}}$.)
Event3. Packet q with the priority larger than $X.p$ (i.e. $X.q > X.p$) enters the server.

We first show for all the flows g with nonempty queue at s, regardless of which event occurs, the following holds:

$$(s \le T.g) \qquad \text{or} \qquad (X.p < X.g). \tag{A2}$$

With Event1, $s = T.g$ since both s and $T.g$ are zero. With Event2, $s = T_{bnd} \le T.g$ from Theorem 1. With Event3, $X.q > X.p$ and $X.q < X.g$ since the queue with the smallest priority value is served first. Thus $X.p < X.g$. Therefore, Eq. (A2) holds at s. Next, we will show that

$$s \le T.p. \tag{A3}$$

With Event1, $s = 0 \le T.p$. With Event2, from Theorem 1, $s = T_{bnd} \le T.p$. With Event3, queue.f must be empty at s. If this is not true, the packet denoted by p' at the head of queue.f at s will have the priority $X.p'$ less than $X.q$ since $X.p' \le X.p$ and $X.p < X.q$. Therefore, the packet entering the server at s is not q but p'. This violates the occurrence of Event3. Thus queue.f must be empty at s. This implies that the packet of flow f arriving at or later than s is with a packet tag no less than s. Since p becomes the head of queue.f at u and queue.f is empty at s, p must not arrive before s. Thus, $T.p \ge s$ and (A3) holds.

Now, consider how long after s packet p will depart from the system. From the definition of s, only the packets r with $X.r \le X.p$ would enter the server after s before p enters the server. Let packets r be from flows h and become the head of queue.h at u'. If we replace p and u with r and u', respectively, s is still the latest time to u' such that either of the three above events occurs. Thus, $T.r \ge s$ from Eq. (A3).

Since $L.r = (X.r - T.r)R.h, s \le T.r$, and $X.r \le X.p$, the total bits of the packet p and all the packets represented by r do not excess $(X.p - s)\sum_{i=0\ldots N-1} R.i$. These bits plus the residual bits of the packet served at s is at most $(X.p - s)C + L_{\max}$ from inequality (2). This amount of data is sent out by the server from s until z, which is equal to $(z - s)C$. Thus, $(z–s)C \le (X.p–s)C + L_{\max}$. Therefore $z \le X.p + L_{\max}/C$.

Theorem 3. BTFQ belongs to the GR class. That is, a packet p of flow f will depart from the BTFQ system by GRC(p) + β, where β is $L_{\max}/C$.

Proof. We know the bound of the departure time of p from Theorem 2, but this time is in TagClock while the time for computing GRC(p) is in real clock. Therefore, consider an alternate BTFQ transferring TagClock to a real clock. Instead of letting the time in TagClock jump forward δ seconds, the alternate BTFQ subtracts δ from each queue tag and priority. Since the relative values of the tags and priorities with respect to each other and with respect to TagClock is the same as in the original BTFQ, the order in which packets are served remains the same. Thus the original and the alternate introduce the same bounded delay.

In the alternate BTFQ, $X.p$ must not excess GRC(p), since the packet priority in the alternate BTFQ can be reduced. Therefore, with Theorem 2, the departure time of packet p in BTFQ does not exceed GRC(p) + $L_{\max}/C$.

Theorem 4. Let flow f be constrained by the Leaky Bucket with parameter ($\sigma.f$, $R.f$) and it passes through K BTFQ servers each of which has capacity C_i, $0 \le i \le K$. The end-to-end delay $\mathrm{D_{E2E}}(f)$, if link propagation delays are not counted in, of the packet of f is bounded as

$$\mathrm{D_{E2E}}(f) \le \sigma.f/R.f + (K-1)L.f_{\max}/R.f. + \sum_{i=1..K} L_{\max}/C_i. \tag{A4}$$

Proof. From Definition 6, a packet p of flow f passing through the GR server (i.e. the server employing a GR scheme) departs from the server by GRC(p) + β, where β is a small constant. With the GR framework [19], if flow f is constrained by the Leaky Bucket ($\sigma.f$, $R.f$) and passes through KGR servers and the packet p departs from each server i by $\mathrm{GRC}_i(p) + \beta_i$, the end-to-end delay of p is bounded as

$$\text{end-to-end delay of } p \le \sigma.f/R.f + (K-1)L.f_{\max}/R.f + \sum_{i=1..K} \beta_i.$$

From Theorem 3 BTFQ is a GR scheme; that is, the packet p departs from the BTFQ server i by $\mathrm{GRC}_i(p) + \beta_i$, where β_{i} is $L_{\max}/C_i$. Therefore, the theorem is proven immediately.

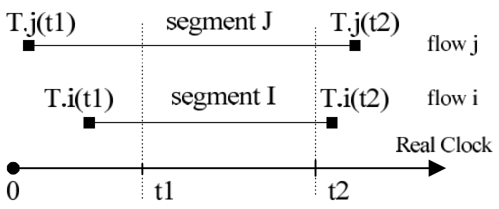


Fig. 14. Relationship of flow tags.

Theorem 5. For the BTFQ system in the current busy period, if current time is t in TagClock and *e(t)* is packet r, then for each flow f backlogged at t the queue tag $T.f$ is bounded as

$$X.r - S_{\max} \le T.f \le X.r + L_{\max}/C. \quad \text{(A5)}$$

Proof. In this proof, time is referred to TagClock. Let p be the packet at the head of queue.f at t. Packet p becomes the head of queue.f at t' ($t' \le t$). Let p' be the packet from f immediately preceding p. Also, let q and q' be the packets in the server at t and t', respectively. And, let r' denote the packet $e(t')$. We depict their relation in Fig. 13.

Since $T.f$ at t is $T.p$, we will show

$$X.r - S_{\max} \le T.p \le X.r + L_{\max}/C. \quad \text{(A6)}$$

From the Enqueue and Dequeue processes, we know the value of $T.p$ is either from TagClock or $X.f$ (not updated) at t'. Therefore, $T.p$ is either t' or $X.p'$. We prove the theorem in the two cases: $T.p = t'$ and $T.p = X.p'$.

Case 1. $T.p = t'$.

C1_Right. Show $T.p \le X.r + L_{max}/C$. Clearly, t' is no greater than the departure time of q'. With Theorem 2, $t' \le X.q' + L_{\max}/C$. By the definition of r, $T.p \le X.r + L_{\max}/C$.

C1_Left. Show $T.p \ge X.r – S_{max}$. Since the value of $T.p$ ($T.p = t'$) is obtained from TagClock, packet p must find its queue being empty when it arrives. The assignment in Enqueue, TagClock $\leftarrow$ max{T_{bnd}, TagClock}, is thus executed. Clearly, $T.p \ge T_{\text{bnd}}$ at t'. Since T_{bnd} is $X.q' - S_{\max}$ at t', consider q' on both cases: $X.q' = X.r'$ and $X.q' < X.r'$.

C1_Left.1. $X.q' = X.r'$. Since $T.p \ge T_{\text{bnd}}$ at t', we have $T.p \ge X.r' – S_{\max}$.

C1_Left.2. $X.q' < X.r'$. Let s be the time q' became the head of its queue. Thus, $s \le t'$. We first show that $s \ge X.r' - S_{\max}$ and $T.q' \ge X.r' - S_{\max}$ and then that $T.p \ge X.r' - S_{\max}$.

Because r' is not q' in this case and q' is in the server at t', r' departs from the server before q'. Since r' is the latest and largest-priority-value packet to time t', immediately before r' enters the server, all queues must be empty except the queue of r'. If there were no packet arriving during the service of r', q' would either belong to another busy period or enter the server after t'. (The reason is as follows. Suppose that no packet arrives during the service of r'. If the queue queue.g where r' is from is empty during this service, q' belongs to another busy period; otherwise, q' arrives after the time t'' when packet r'' enters the server. Here, r'' is the packet at the head of queue.g during the service of r'. We have that $t'' > t'$, since $e(t'') = r''$, $e(t') = r'$, and $X.r'' > X.r'$. Thus q' arrives after t'. Clearly, q' enter the server after t'.) Either of the cases violates the proposition that r' and q' is in the same busy period and q' is in the server at t'. Therefore, there must be packets arriving during the service of r'. Let packet q'' be the first one arriving. Once q'' arrives, the assignment "TagClock $\leftarrow$ max{T_{bnd}, TagClock}" in Enqueue is executed. Thus the arrival time of q'' is no less than the T_{bnd} which is $X.r' - S_{\max}$ at this time. Because q' either is q'' or arrives after q'', the time s (when q' becomes the head of its queue) must be not earlier than the arrival time of q''. Thus, $s \ge X.r' – S_{\max}$. Since the queue of q' is empty when q'' arrives, $T.q'$ is not less than the T_{bnd} (i.e. $X.r' - S_{\max}$) at the arrival of q''. Thus $T.q' \ge X.r' - S_{\max}$. Since $T.p = t' \ge s$, $T.p \ge X.r' - S_{max}$.

From C1_Left.1 and C1_Left.2, we conclude $T.p \ge X.r' - S_{\max}$. Clearly, $X.r$ must be no less than $X.r'$. If $r = r'$, we are done. Otherwise, r must enter the server after t' and not later than t. Since p becomes the head of queue.f at t' and is not in the server at t, we have $X.p \ge X.r$. Since $L.p/R.f \le S_{\max}$, $T.p \ge X.r - S_{\max}$.

*Case*2. $T.p = X.p'$.

C2_Right. Show $T.p \le X.r + L_{\max}/C$. Suppose $T.p > X.r + L_{\max}/C$. Since $X.p' = T.p > X.r$, p' is not r. Packet p' enters the server by t', since p becomes the head of queue.f at t'. This violates the definition of r. So, $T.p \le X.r + L_{max}/C$.

C2_Left. Show $T.p \ge X.r - S_{\max}$. At time t' (when p becomes the head of queue.f), there are two cases: One is that p' enters the server, so $p' = q'$; another is that p arrives and finds queue.f being empty, so $p' \ne q'$.

C2_Left.1. $p' = q'$. We have $T.p = X.q'$. Because $X.q' \le X.r'$, consider $X.q' = X.r'$ and $X.q' < X.r'$. If $X.q' = X.r'$, then $T.p = X.r'$. So, $T.p \ge X.r' - S_{max}$. On the other hand, consider $X.q' < X.r'$. Let s be the time when q' becomes the head of its queue. Thus, $s \le t'$. With the same argument as C1_Left.2, we have $s \ge X.r' - S_{\max}$ and $T.q' \ge X.r' - S_{\max}$. Since $p' = q'$, we have $T.p = X.p' = X.q' > X.r' - S_{max}$.

C2_Left.2. $p' \ne q'$. We have $T.p = X.p' \ge$ TagClock ($= t'$) $\ge T_{\text{bnd}}$ at t'. If $X.q' = X.r'$, $T.p \ge T_{\text{bnd}}$ (at t') $= X.r' – S_{max}$. On the other hand, consider $X.q' < X.r'$. Let s be the time when q' becomes the head of its queue. Thus, $s \le t'$. With the same argument as C1_Left.2, we have $s \ge X.r' - S_{max}$ and $T.q' \ge X.r' - S_{max}$. Since $T.p = X.p' \ge t'$, so $T.p \ge s \ge X.r' – S_{\max}$.

From C2_Left.1 and C2_Left.2, we conclude $T.p \ge X.r' - S_{\max}$. As in the conclusion of C1_Left, we have $T.p \ge X.r - S_{\max}$. Therefore, with Case1 and Case2, this theorem is proven.

Theorem 6. For the BTFQ scheme, the fairness

$F.ij(t1, t2)$ between any two flows i and j backlogged during time interval $[t1,t2]$ is bounded as

$$F.ij(t1, t2) \leq S_{\max} + \max\{L.i_{\max}/R.i, L.j_{\max}/R.j\} + L_{\max}/C. \tag{A7}$$

Proof. Since $F.ij(t1, t2)$ is defined as $|W.i(t1, t2)/R.i - W.j(t1, t2)/R.j|$, consider $W.f(t1, t2)$ first, where f is i or j. Because f is backlogged during $[t1,t2]$, $T.f$ advances step by step during $[t1,t2]$. Let $T.f(t)$ be $T.f$ at time t. Define the function $\pi.f(t)$ for flow f as follows. $\pi.f(t)$ is the residual amount of the packet p in the server at t, if p is from f; otherwise, zero. Therefore,

$$W.f(t1, t2) = R.f^*(T.f(t2) - T.f(t1)) + \pi.f(t1) - \pi.f(t2) \tag{A8}$$

$$W.f(t1, t2)/R.f = (T.f(t2) - \pi.f(t2)/R.f) - (T.f(t1) - \pi.f(t1)/R.f) \tag{A9}$$

Fig. 14 shows the relationship of $T.i(t1)$, $T.j(t1)$, $T.i(t2)$, and $T.j(t2)$, where the length of segment I and J represents the value of $W.i(t1, t2)/R.i$ and $W.j(t1, t2)/R.j$, respectively, if i and j are not served at $t1$ and $t2$.

Let $w.ji$ denote $(W.j(t1, t2)/R.j - W.i(t1, t2)/R.i)$. Then, $F.ij(t1, t2) = \max\{w.ji, w.ij\}$. To find the upper bound of $F.ij(t1, t2)$ is to find the upper bounds of $w.ji$ and $w.ij$. We first consider $w.ji$. From the Fig. 14, the maximum of $w.ji$ occurs when segment J and I is as long as and as short as possible at the same time, respectively. From (A9), $w.ji$ can be written as $w.ji = M1 + M2$, where

$$M1 = [T.i(t1) - T.j(t1) + \pi.j(t1)/R.j - \pi.i(t1)/R.i],$$

$$M2 = [T.j(t2) - T.i(t2) + \pi.i(t2)/R.i - \pi.j(t2)/R.j].$$

The work of finding the upper bound of $w.ji$ is decomposed into two sub-works as follows:

Work1 is to find the upper bound of $M1$. Consider the flow the packet in the server at $t1$ belongs to. We have the following cases.

W1_Case1. The packet, denoted by p, in the server at $t1$ belongs to flow i. Thus, $\pi.i(t1) \geq 0$ and $\pi.j(t1) = 0$. From Theorem 1, $T.j(t1) \geq T_{\text{bnd}}$ (at $t1$) $= X.p - S_{\max}$. Since i is backlogged at $t1$, $T.i(t1) = X.p$. Therefore, $M1 \leq S_{max}$.

W1_Case2. The packet, denoted by q, in the server at $t1$ belongs to flow j. Thus, $\pi.j(t1) \leq L.q$ and $\pi.i(t1) = 0$. With Definition 7, let packet r be $e(\tau 1)$, where $\tau 1$ is the value of TagClock at $t1$. From Theorem 5, $T.i(t1) \leq X.r + L_{\max}/C$. From the definition of r, $X.q \leq X.r$. If $X.q = X.r$, $T.j(t1) = X.r$. Thus, $M1 \leq L.q/R.j + L_{\max}/C$. If $X.q < X.r$, from "C1_Left.2" in the proof of Theorem 5, $T.q \geq X.r - S_{max}$. Since $X.q = T.j(t1)$, $T.j(t1) \geq X.r - S_{max} + L.q/R.j$. Hence, $M1 \leq S_{max} + L_{max}/C$.

W1_Case3. The packet in the server at $t1$ belongs to flow k, where $k \neq i, j$. Thus $\pi.j(t1)$ and $\pi.i(t1)$ are zero. From Theorem 5, $T.i(t1) - T.j(t1) \leq S_{\max} + L_{\max}/C$. Hence, $M1 \leq S_{\max} + L_{\max}/C$. From W1_Case1, 2, and 3, we conclude $M1 \leq S_{\max} + L_{\max}/C$.

Work2 is to find the upper bound of $M2$. Consider the following cases.

W2_Case1. If flow j is never served during $[t1, t2]$, we have $w.ji \leq 0$ since $W.j(t1, t2) = 0$.

W2_Case2. Consider that flow j is served during $[t1, t2]$ and the packet, denoted by p, in the server at $t2$ is from flow i. We have $\pi.j(t2) = 0$, $\pi.i(t2) \leq L.p$, and $T.i(t2) = X.p$ since flow i is backlogged at $t2$. Let q be the latest packet of flow j entering the server no later than $t2$. We have $T.j(t2) = X.q$. Consider $X.q \leq X.p$. We have $T.j(t2) \leq T.i(t2)$. Since $\pi.i(t2) \leq L.p$ and $\pi.j(t2) = 0$, $M2 \leq L.p/R.i \leq L.i_{\max}/R.i$. On the contrary, consider $X.q > X.p$. Let s be the latest time, no later than $t2$, such that flow i becomes backlogged. Let u be the time when q enters the server. We have $s \geq u$, since p enters the server after q and $X.q > X.p$. Because i is backlogged during $[t1, t2]$, $s \leq t1 \leq t2$. Thus $u \leq t1 \leq t2$. $T.j(t1) = T.j(t2) = X.q$, since q is the latest packet entering the server no later than $t2$. From (A9), $W.j(t1, t2)/R.j = \pi.j(t1)/R.j - \pi.j(t2)/R.j \leq L.j_{max}/R.j$. Since $W.i(t1, t2)/R.j \geq 0$, $w.ji \leq L.j_{max}/R.j \leq S_{max}$.

W2_Case3. Consider that flow j is served during $[t1, t2]$ and the packet in the server at $t2$ is not from flow i. We have $\pi.i(t2) = 0$. Let p be the packet at the head of queue.i at $t2$. Let q be the latest packet of flow j entering the server no later than $t2$. Thus $T.p = T.i(t2)$ and $X.q = T.j(t2)$. Consider $X.q \leq X.p$. We have $T.j(t2) \leq T.i(t2) + L.p/R.i$. Since $\pi.j(t2) \geq 0$ and $\pi.i(t2) = 0$, $M2 \leq L.p/R.i \leq L.i_{\max}/R.i$. On the contrary, consider $X.q > X.p$. Let s be the latest time, no later than $t2$, such that flow i becomes backlogged. Let u be the time when q enters the server. We have $s \geq u$, since p enters the server after q and $X.q > X.p$. Thus $u \leq s \leq t1 \leq t2$. This implies $T.j(t1) = T.j(t2) = X.q.W.j(t1, t2)/R.j \leq L.j_{\max}/R.j$. Thus, $w.ji \leq L.j_{\max}/R.j \leq S_{\max}$.

From Work1 and Work2, $w.ji = M1 + M2 \leq S_{\max} + L.i_{\max}/R.i + L_{max}/C$. We find the upper bound of $w.ij$ in the same way as above. Thus, $w.ij \leq S_{max} + L.j_{max}/R.j + L_{max}/C$. Since $F.ij(t1, t2)$ is $\max\{w.ij, w.ji\}$, $F.ij(t1, t2) \leq S_{max} + \max\{L.i_{\max}/R.i, L.j_{\max}/R.j\} + L_{\max}/C$.

References

[1] M. de Prycker, Asynchronous Transfer Mode: Solution for Broadband ISDN, 2, Ellis Horwood, New York, 1993.

[2] J. Kurose, Open issues and challenges in providing quality of service guarantees in high speed networks, ACM Computer Communication Review 23 (1) (1993) 6–15.

[3] L. Zhang, Virtual Clock: a new traffic control algorithm for packet switching, ACM Transactions on Computer Systems 9 (2) (1991) 101–124.

[4] A.K. Parekh, R.G. Gallager, Generalized processor sharing approach

to flow control in integrated services networks-the single node case, IEEE INFOCOM (1992) 915–924.

[5] A.K. Parekh, R.G. Gallager, Generalized processor sharing approach to flow control in integrated services networks—the multiple node case, IEEE INFOCOM (1993) 521–530.

[6] A. Demers, S. Keshav, S. Shenkar, Analysis and simulation of a fair queueing algorithm, Internetworking: Research & Experience 1 (1) (1990) 3–26.

[7] D.A. Pucknell, K. Eshraghian, Basic VLSI Design: Systems and Circuits, 2, Prentice Hall, New York, 1987.

[8] S.J. Golestani, A self-clocked fair queueing scheme for broadband applications, IEEE INFOCOM (1994) 636–646.

[9] P. van Emde Boas, R. Kaas, E. Zijlstra, Design and implementation of an efficient priority queue, Mathematical System Theory 10 (2) (1977) 99–127.

[10] W.C. Hon, D.H.K. Tsang, Y. Tao, Bandwidth allocation for VBR video traffic in ATM networks, Fourth International Conference on Computer Communications and Networks, 1995, pp. 612–615.

[11] P. Goyal, H.M. Vin, H. Cheng, Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks, IEEE/ACM Transactions on Networking 5 (5) (1997) 690–704.

[12] S. Suri, G. Varghese, G. Chandranmenon, Leap forward virtual clock: a new fair queuing scheme with guaranteed delay and throughput fairness, IEEE INFOCOM (1997) 557–565.

[13] Y.P. Chu, E.H. Hwang, A new packet scheduling algorithm: minimum starting-tag fair queueing, IEICE Transactions on Communications E80-B (10) (1997) 1529–1536.

[14] J. Cobb, M. Gouda, A. El-Nahas, Time-shift scheduling: fair scheduling of flows in high-speed networks, IEEE International Conference on Network Protocols, 1996, pp. 6–13.

[15] D. Stiliadis, A. Varma, Frame-based fair queueing: a new traffic scheduling algorithm for packet switched network, ACM SIGMETRICS conference on measurement and modeling of computer systems, 1996, pp. 104–115.

[16] J. Turner, New directions in communications (or which way to the information age?), IEEE Communications Magazine 24 (10) (1986) 8–15.

[17] V.C. Hamacher, Z.G. Vranesic, S.G. Zaky, Computer Organization, McGraw-Hill, New York, 1990, pp. 274–276.

[18] G.G. Xie, S.S. Lam, Delay guarantee of virtual clock server, IEEE/ACM Transactions on Networking 3 (6) (1995) 683–689.

[19] P. Goyal, S.S. Lam, H.M. Vin, Determining end-to-end delay bounds in heterogeneous networks, International Workshop on Network and Operating System Support for Digital Audio and Video, 1995, pp. 287–298.

[20] D. Stiliadis, A. Varma, Rate-Proportional Servers: a design methodology for fair queueing algorithms, IEEE/ACM Transactions on Networking 6 (2) (1998) 164–174.

[21] D. Stiliadis, A. Varma, Efficient fair queueing algorithms for packet-switched networks, IEEE/ACM Transactions on Networking 6 (2) (1998) 175–185.

[22] W.E. Leland, M.S. Taqqu, W. Willinger, D.V. Wilson, On the self-similar nature of ethernet traffic (extended version), IEEE/ACM Transactions on Networking 2 (1) (1994) 1–15.

[23] S. Bates, S. McLaughlin, Testing the Gaussian assumption for self-similar teletraffic models, IEEE Signal Processing Workshop on Higher-Order Statistics, 1997, pp. 444–447.