PAPER

Packet-Mode Scheduling with Proportional Fairness for Input-Queued Switches

Kang XI[†a)], ***Nonmember***, **Shin'ichi ARAKAWA**[††], **Masayuki MURATA**[†††], ***Members***,
Ning GE[††††], ***and*** **Chongxi FENG**[††††], ***Nonmembers***

SUMMARY Proportional fair bandwidth allocation in packet switches is a fundamental issue for quality of service (QoS) support in IP networks. Input-queued switches performing packet-mode scheduling deliver all the segments of a packet contiguously from the input port to the output port, thus greatly simplify the output reassembly and yield performance advantage over switches with cell-mode scheduling under certain conditions[1]. An important issue of packet-mode scheduling is how to achieve fair bandwidth allocation among flows with different packet sizes. This paper presents a packet-mode fair scheduling (pFS) algorithm to guarantee each flow a bandwidth proportional to its reserved share regardless of the packet size distribution and the system load. Simulations show that our approach achieves high throughput as well as good delay performance. Compared to algorithms without fairness mechanism, pFS yields significant performance improvement in terms of average packet delay under heterogeneous traffic. An hardware implementation is presented to show that the computation of the algorithm can be completed within a single clock, which makes pFS applicable to high speed switches.

key words: *scheduling, fairness, switch, input-queued, packet-mode.*

1. Introduction

Input-queued switches are able to overcome head-of-line (HOL) blocking and achieve high throughput by using virtual output queueing (VOQ) and efficient scheduling algorithms. In this case, the buffers and control logic work at the line speed [2], which enables such architecture to be scaled up to backbone routers with very high speed, as realized in Tiny Tera [3], Cisco GSR [4], Lucent GRF [5], etc.

Although Internet Protocol (IP) packets have variable lengths, switches are usually operated in fixed-size cells internally. Figure 1 shows a typical architecture, where IP packets are segmented into cells in the input ports, switched in the crossbar, and reassembled in the output ports. Cell delivery in the crossbar is controlled by a scheduler that calculates a bipartite match from the input ports to the output ones according to the backlog in the VOQs. If all the input-output matches are recalculated in each slot, the algorithm is called cell-mode scheduling. In this case, cells from different input ports may interleaved in an output port, thus reassembly modules are needed. On the other hand, if an input-output match is kept until all the cells of a packet are delivered, the algorithm is called packet-mode scheduling. In this case, cells belonging to the same packet arrive at the output port contiguously, thus the reassembly is greatly simplified with savings in both complexity and memory. It has been shown that packet-mode scheduling achieves 100% throughput and yields performance advantage over cell-mode scheduling in terms of packet delay under certain conditions [1][6].

To support quality of service (QoS) in IP networks, schedulers are required to perform proportional fair bandwidth allocation among competing flows, which means: first, a flow with arrival rate less than its reservation is fully served; second, excess bandwidth (not used up by light load flows) is allocated among heavy load flows proportionally

[†]The author was with the Graduate School of Information Science and Technology, Osaka University, Machikaneyama, Toyonaka, Osaka 560–0043, Japan, and is currently with the Dept. of ECE, Polytechnic University, Brooklyn, NY 11201, USA.
[†††]The author is with the Graduate School of Information Science and Technology, Osaka University, Machikaneyama, Toyonaka, Osaka 560–0043, Japan.
[††]The author is with the Graduate School of Economics, Osaka University, Machikaneyama, Toyonaka, Osaka 560–0043, Japan.
[††††]The authors are with the Department of Electronic Engineering, Tsinghua University, Haidian, Beijing, 100084, China
a) E-mail: kangxi@ieee.org, Fax: 6-6850-6868

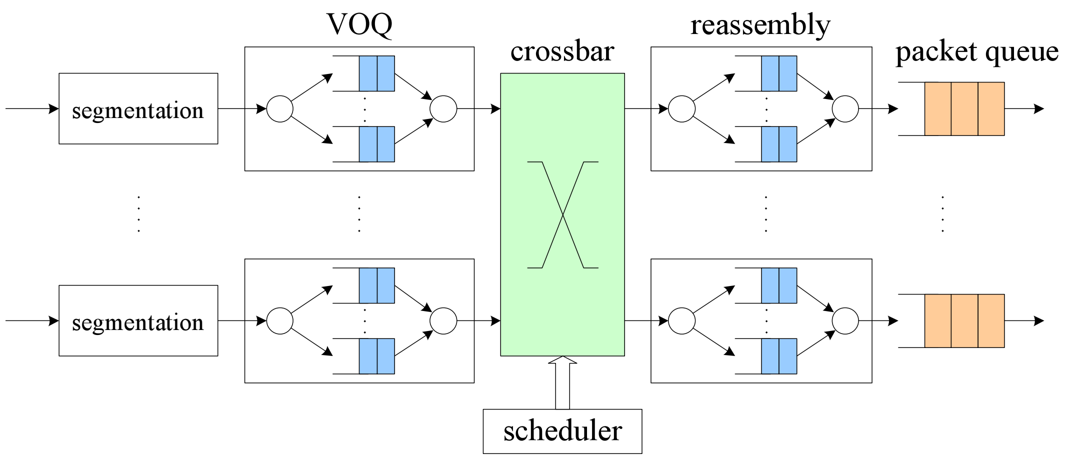

Fig. 1 Architecture of input-queued packet switch.

to their reserved shares [7]. Fairness mechanism is also important in best-effort networks where misbehaving flows should be prevented from grabbing too much bandwidth so as to protect normal ones. Although fair scheduling has been intensively investigated in output-queued switches, it is non-trivial for input-queued architecture to achieve good fairness and high throughput simultaneously since a bipartite match conforming the bandwidth regulation may not guarantee throughput performance, and vice versa [8].

This paper proposes a packet-mode fair scheduling (pFS) that guarantees proportional fairness. With a bandwidth-sensitive matching policy, the algorithm controls the setup of each input-output match pair according to its bandwidth usage, such that good fairness is achieved. Besides good fairness, pFS also achieves high throughput and low delay. In particular, simulations show that our algorithm yields significant performance improvement over algorithms without fairness mechanism under heterogeneous traffic, which is of special importance to real networks carrying diverse application data. A hardware implementation of pFS is presented to show that the fairness-related computations can be completed within a single clock cycle, which makes our algorithm applicable to high speed switches with strict requirement on complexity [9][10].

The remainder of this paper is organized as follows. Section 2 gives a brief review of the background and related works. Section 3 describes the proposed algorithm in detail. Performance evaluation of the algorithm is given in Section 4, and the hardware implementation is presented in Section 5. Finally, Section 6 concludes the paper.

Throughout this paper, a *cell* stands for a small data flit with fixed size, while a *packet* has variable size and can be segmented into multiple cells.

2. Background and Related Work

2.1 Cell-Mode and Packet-Mode Scheduling

Since switches are operated in cells internally, the time axis can be divided into fixed-length cell slots, where each slot allows at most one cell to be delivered from/to an input/output port. Note that an input may have cells destined to multiple outputs and an output may be competed by multiple inputs, a scheduler is needed to find a bipartite match that establishes one-to-one match pairs from inputs to outputs.

A cell-mode scheduler releases *all the match pairs* and performs recalculation in *each slot*. This type of scheduling has been intensively researched in literature, such as iSLIP [11], oldest cell first (OCF), longest queue first (LQF) [12][13], parallel iterative matching (PIM) [14], and dual round-robin matching (DRRM) [15][16]. Since cells from different packets may be interleaved in output ports, cell-mode scheduling introduces additional complexity of packet reassembly in output ports when it is applied to packet switches.

Packet-mode schedulers differ from cell-mode ones in that a match pair is *kept unchanged* until all the cells belonging to the same packet are delivered. In each slot, only the idle unmatched ports and those that just delivered a complete packet in the previous slot are taken for match recalculation [1][17]. This mechanism guarantees that cells belonging to a single packet arrive at the destination output port contiguously, thus greatly facilitates output assembly. It has been shown that this type of scheduling achieves 100% throughput and yields low average delay [1]. The properties of low complexity and good performance make packet-mode scheduling a strong candidate for high speed IP routers.

2.2 Fair Scheduling in Input-Queued Switches

Existing research on fair scheduling mainly focuses on output-queued switches, where only the output ports experience contention. Several algorithms have been proposed, such as generalized processor sharing (GPS) [18], deficit round-robin (DRR) [19] and elastic round-robin (ERR) [20]. Input-queued switches differ from output-queued ones in that contention takes place at both input and output ports, thus makes it difficult to achieve high throughput and good fairness simultaneously. Most existing algorithms are designed for cell-mode schedulers. Weighted PIM (WPIM) employs weight-based masks in the matching process and guarantees the reserved bandwidth of each flow [21]. However, the excess bandwidth are allocated equally (rather than proportionally) among the competing flows. Based on GPS, iterative fair scheduling (iFS) derives a virtual time stamp for each arriving cell and calculates bipartite match according to the time stamps of the HOL packets in the VOQs [8]. It is well know that GPS-based algorithms suffer from the complexity of time stamp calculation and sorting [14], which makes iFS unsuitable for high speed switches.

Fairness mechanism is of special importance to packet-mode scheduling where the lifetime of each match pair equals to the size of the packet being delivered. In case of congestion, it is likely that flows with large packets get more bandwidth than those with small packets, thus makes fairness mechanism a fundamental issue for packet-mode schedulers. However, cell-mode fairness algorithms cannot be easily extended to cover this issue and existing research is still limited. Reference [22] studies the problem of delay bounds but does not address proportional fairness. An algorithm called iterative DDR (iDDR) is presented in [23] to provide fair bandwidth allocation, yet the output of DRR is bursty [20], which may cause potential performance degradation in the downstream nodes.

3. Packet-Mode Fair Scheduling

3.1 Framework

This paper considers bandwidth allocation for coarse grain flows and leaves fine grain control to line cards before packets are segmented and put into VOQs. The concepts of flow and proportional fairness in this paper are defined in the following:

Definition 1: In an $N \times N$ switch, flow $F_{i,j}$ is defined to be the sequence of packets from input i to output j, where $i, j = 0, 1, \ldots N-1$.

Definition 2: Denote the arrival rate of $F_{i,j}$ with $a_{i,j}$, let $b_{i,j}$ and $b^*_{i,j}$ stand for the reserved and allocated bandwidth of $F_{i,j}$, respectively, the proportional fairness factor of the scheduling algorithm is defined as

$$f = \max_{i,j,m,n} \left(\left| \frac{b^*_{i,j}}{b_{i,j}} - \frac{b^*_{m,n}}{b_{m,n}} \right| e_{i,j} e_{m,n} \right), \tag{1}$$

where

$$e_{i,j} = \begin{cases} 0 & \text{if } a_{i,j} \le b^*_{i,j} \\ 1 & \text{otherwise} \end{cases}, \quad i, j = 0, \ldots, N-1. \tag{2}$$

Since the aggregated reservation of a link never exceeds its capacity, $f = 0$ indicates that the light load flows ($a_{i,j} \le b^*_{i,j}$) are fully served and the excess bandwidth (if non-zero) is allocated among the heavy load ones ($a_{i,j} > b^*_{i,j}$) proportionally to their reservations.

The design object of pFS is to minimize f as well as to keep high throughput and low delay. Our algorithm is based on three steps: *request* $\rightarrow$ *grant* $\rightarrow$ *accept*, where fairness mechanism is introduced in the grant and the accept steps to control the bipartite matching. The grant arbiter G_j in output j and the accept arbiter A_i in input i have the same structure and maintain pointers to indicate port index from 0 to $N-1$ (details of the arbiter will be given in Section 3.2). Denote the pointers of G_j and A_i with g_j and a_i, respectively, the three steps of pFS include:

request : Each unmatched input i sends a request to every output j if the queue of $F_{i,j}$ is non-empty;

grant : If output j is unmatched, G_j is activated to select an input from the requests as the grant. If the grant is accepted in the next step, pointer g_j is increased one location beyond the selected input.

accept : Upon the grants to input i, A_i is used to select an output to generate a match pair, and a_i is increased one location beyond the accepted output.

Once a match pair is established, it will be kept unchanged until a complete packet is delivered. In iterative algorithm, the three steps can be repeated multiple times to improve throughput. Similar to iSLIP, iterative pFS updates a_i and g_j in the first iteration to avoid port starvation, this can be briefly explained as follows: Since the pointers are moved in a round-robin way, each granted input becomes the lowest priority while the next input gets the highest priority. Keeping the pointers unchanged other than the first iteration makes an output continue to grant the highest priority input until it is successful, thus no input queue will be skipped infinitely. An example and detailed explanation can be found in [11].

3.2 Credit-Based Arbiter

Since the grant and accept arbiters have the same structure, we use a general model to show the principle, as illustrated in Fig. 2. Boolean inputs $in_0, \ldots, in_{N-1}$ stand for the requests/grants in a grant/accept arbiter, and the signal *out* indicates the selected input/output port. To control the bandwidth usage of $F_{i,j}$ in both input i and output j, A_i maintains a credit $c^A_{i,j}$ and G_j maintains $c^G_{i,j}$, the notation is simplified to c_i in Fig. 2.

Internally each arbiter consists of two selectors: master and slave. Request from a flow with exhausted credit is filtered out from the master selector, while the slave accepts all the requests. Both the master and the slave start from the same pointer p to find the first request in a round-robin way, and the result from the slave selector is adopted only when the master output is *null*, which means the all the requesting flows have exhausted their credits. By adjusting the credit of each flow appropriately, the master selector is able to achieve fair bandwidth allocation since misbehaving flows will be filtered out. However, only considering flows with surplus credits does not guarantee high throughput since a maximal bipartite match may not be found out, thus slave selectors are introduced to work as a supplementary approach by filling the vacancies in the bipartite match in case the master selectors cannot find eligible candidates. It is worth noting that this operation does not affect the fairness since the over-subscription is always recorded in c_i and such flows will get punished at a later time. The arbitration algorithm is explained in pseudo code below:

```
Algorithm Arbitration Process
1.  $out^m \leftarrow$null;  $out^s \leftarrow$null;
2.  for $n \leftarrow 0$ to $N-1$
3.          $i \leftarrow (n+p) \mod N$;
4.          if $in_i$ =true and $c_i > 0$
5.            then $out^m \leftarrow i$; break;
6.  for $n \leftarrow 0$ to $N-1$
7.          $i \leftarrow (n+p) \mod N$;
```

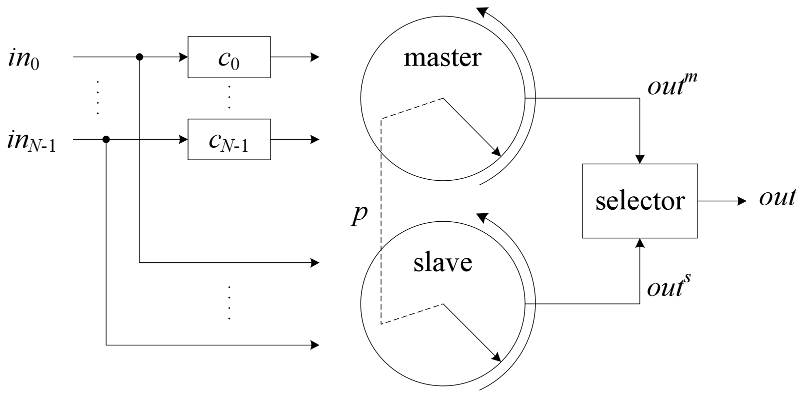

Fig. 2 Architecture of a grant/accept arbiter.

```
8.          if $in_i$ =true
9.             then $out^s \leftarrow i$; break;
10. if $out^m$ =null
11.    then $out \leftarrow out^s$;
12.    else $out \leftarrow out^m$;
13. $p \leftarrow (out + 1) \mod N$;
```

3.3 Credit Management

Since the credit management of an arbiter is independent to the others, we choose the following notations to simplify the descriptions of a single arbiter: the flows under control are F_i ($i = 0, \ldots, N-1$), the bandwidth reservation and the credit of F_i are b_i and c_i, respectively.

At slot n, each flow F_i is classified into three states according to its queue length $Q_i(n)$ and credit $c_i(n)$:

active :	$Q_i(n) > 0$;
waking :	$Q_i(n) = 0$ and $c_i(n) \leq 0$;
idle :	$Q_i(n) = 0$ and $c_i(n) > 0$.

To achieve proportional fairness, each arbiter only increases the credits of *active* and *waking* flows while keeps the credits of *idle* flows unchanged. The credits are updated as following:

$$c_i(n+1) = c_i(n) - s_i(n) + \frac{b_i}{\sum_i b_i h_i(n)}, \quad i = 0, \ldots, N-1, \tag{3}$$

where $s_i(n)$ is the number of cell (0 or 1) delivered for F_i in slot n, and $h_i(n)$ is a binary variable indicating the state of F_i:

$$h_i(n) = \left\{ \begin{array}{ll} 1 & \text{if } F_i \text{ is } \textit{active} \text{ or } \textit{waking} \\ 0 & \text{if } F_i \text{ is } \textit{idle} \end{array} \right. , \quad i = 0, \ldots, N-1. \tag{4}$$

The operation of (3) not only guarantees the reserved bandwidth of each flow but also allocates excess bandwidth among the competitors proportionally to their reservations. Note that the non-integer increase of credit in (3) is not applicable to hardware implementation, we modify the process so that credits are updated with integers frame-by-frame, where the frame length is dynamically adjusted according to the system state:

1. Set a frame counter $FC(n)$ that updates as following:

$$FC(n+1) = \left\{ \begin{array}{ll} FC(n) - sent(n) & \text{if } FC(n) \neq 1 \\ \sum_i d_i h_i(n) & \text{if } FC(n) = 1 \end{array} \right. , \quad i = 0, \ldots, N-1, \tag{5}$$

where $sent(n) = \sum_{i=0}^{N-1} s_i(n)$ indicates whether a cell transmission has occurred in slot n and d_i is the integer reservation factor calculated from b_i:

$$\frac{d_i}{\sum_{k=1}^{N} d_k} \approx \frac{b_i}{\sum_{k=1}^{N} b_k}. \quad i = 0, \ldots, N-1, \tag{6}$$

The reason we let FC count from $\sum_i d_i h_i(n)$ to 1 rather than from $\sum_i d_i h_i(n) - 1$ to 0 is to facilitate the hardware implementation, which will be explained in Section 5.2.

2. The credit of a flow is decreased when a cell is delivered, and is increased at the end of a frame ($FC(n) = 1$) only when the flow is *active* or *waking* , thus (3) can be emulated with

$$c_i(n+1) = \begin{cases} c_i(n) - s_i(n) & \text{if } FC(n) \neq 1 \\ c_i(n) - s_i(n) + d_i & \text{if } FC(n) = 1 \end{cases}, \quad i = 0, \ldots, N-1. \tag{7}$$

Note that the frame lengths are calculated from the *active* and *waking* flows and only the credits of such flows are increased, the total service received by each flow is proportional to its reservation, thus the unused bandwidth from light load flows are allocated among heavy load flows with proportional fairness. This can be explained with the following example: Consider three competing flows F_a, F_b and F_c with bandwidth reservation factors 1, 2 and 3, respectively. If all the three queues are full, the frame length is 6 and the bandwidth allocated to the flows are 1/6, 1/3 and 1/2, respectively. In another case, suppose F_a is *idle* and the queues of F_b and F_c are full, then the frame length will be 5 and only the credits of F_b and F_c are increased in each frame, the bandwidth allocated to the three flows change to 0, 2/5 and 3/5, respectively, which means the unused bandwidth from F_a is allocated between F_b and F_c proportionally to their reservations.

Although the state of a flow may change within a frame to cause mismatch with the credit (i.e., an *idle* flow becomes *active* within a frame and may experience credit shortage in that short period), such kind of mismatch will not greatly degrade the throughput performance since the flow may still get service due to the salve selectors if there is no other eligible flows. The negative credit caused by this service will be compensated immediately in the next frame, thus the fairness is still maintained. In addition, (5) shows that the frame counter decreases only when there is a cell transmission, this further improves fairness since the total credit increase matches the total cell departure exactly and no flow accumulates excessive credit during link idle. It is worth noting that WPIM also performs frame-based credit management, however, the frames are with fixed lengths, thus the allocation of the excess bandwidth does not have proportional fairness.

3.4 Discussion

1. By introducing bandwidth-sensitive grant and accept arbiters, the frequency for a match pair to appear is regulated by the resource usage, thus realizes fairness. This can be illustrated using an example with a 2×2 switch where all the packets are 1-cell long (Fig. 3). Suppose the link bandwidth is 1 and the flow bandwidth reservations are $b_{0,0} = 1/3$, $b_{0,1} = 2/3$, $b_{1,0} = 2/3$, and $b_{1,1} = 1/3$. From (6), we have $d_{0,0} = 1$, $d_{0,1} = 2$, $d_{1,0} = 2$ and $d_{1,1} = 1$, respectively.
2. Assume that all the flows are greedy with non-empty queues, using pFS , the update of the credits and the match patterns turn to be periodic: with two *cross* patterns and one *parallel* pattern every 3 slots, as shown in Fig. 4. This guarantees the expected fairness. The results with random traffic load will be demonstrated in Section 4.
3. The fairness of pFS is irrelevant to the packet size of a flow. Since the credit of a flow is decreased according to the number of cells being delivered, the bandwidth control is carried out according to the real amount of bits, thus a flow with large packets does not get more bandwidth than a flow with small ones.

4. Performance Evaluation

4.1 Traffic Scenarios

Packet flows for the simulation are generated with ON-OFF models, where an ON period generates a packet

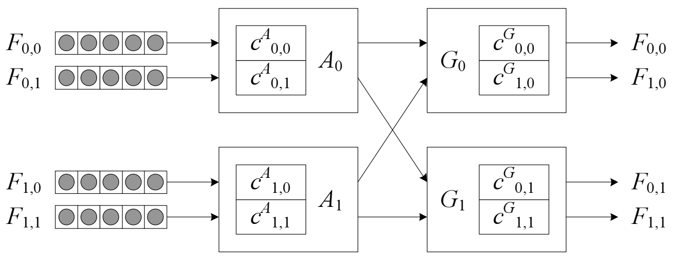

Fig. 3 2×2 switch with greedy flows.

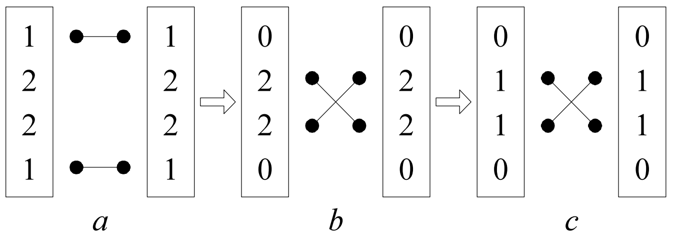

Fig. 4 Credits and match patterns.

with variable size and an OFF period keeps idle. The time unit is based on cell slot, where each cell contains 64 bytes. The length of the OFF period in cell slot is geometrically distributed as

$$P(L_{OFF} = m) = \beta^m(1-\beta), \quad m = 0, 1, \ldots, \tag{8}$$

$$E(L_{OFF}) = \frac{\beta}{(1-\beta)}, \tag{9}$$

where $\beta \in [0,\ 1)$.

The packet size is determined by the length of the ON period. The simulation adopts three types of packet size distribution:

1. $T_A(L)$ is with fixed packet size L:

$$P(L_{ON} = L) = 1, \tag{10}$$

$$E(L_{ON}) = L. \tag{11}$$

2. $T_B(L)$ corresponds to uniform distribution:

$$P(L_{ON} = n) = \frac{1}{L}, \quad n \in [1,\ L], \tag{12}$$

$$E(L_{ON}) = \frac{1+L}{2}. \tag{13}$$

3. T_C is an approximation of the backbone IP packet size distribution, which comes from a 5-minute trace in MCI backbone [24]:

$$P(L_{ON} = n) = \begin{cases} 0.500 & n = 1 \\ 0.033 & n \in [2, 7] \\ 0.070 & n \in [8, 9] \\ 0.004 & n \in [10, 23] \\ 0.100 & n = 24 \end{cases}, \tag{14}$$

$$E(L_{ON}) = 6.28. \tag{15}$$

Denote the load of flow $F_{i,j}$ with $\lambda_{i,j}$. Once the traffic pattern is determined, $\lambda_{i,j}$ can be controlled by β:

$$\lambda_{i,j} = \frac{(1-\beta)E(L_{ON})}{\beta + (1-\beta)E(L_{ON})}, \quad i, j = 0, \ldots, N-1. \tag{16}$$

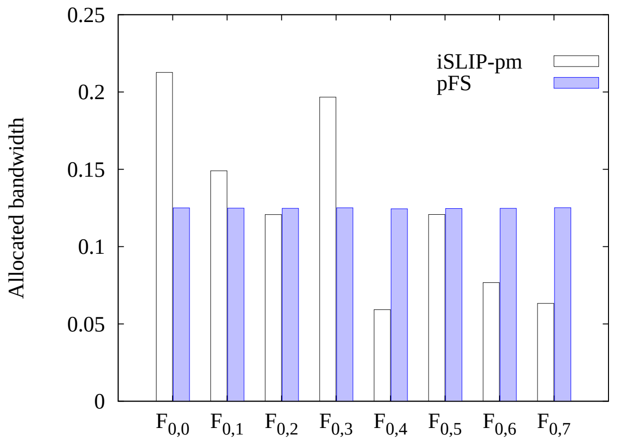

Fig. 5 Allocated bandwidth of $F_{i,0}$.

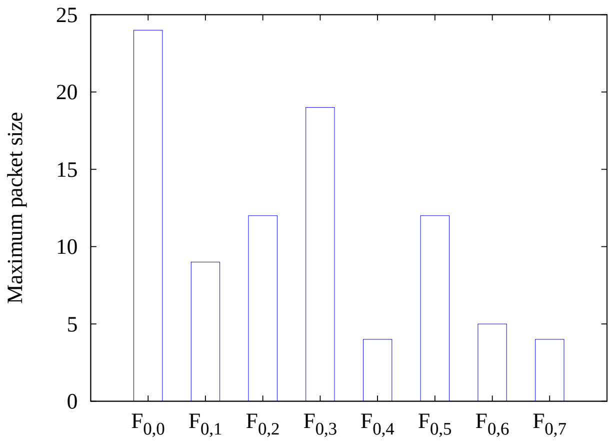

Fig. 6 Maximum packet size of $F_{i,0}$.

4.2 The Relation between Fairness and Packet Size

It has been explained in Section 3.4 that the fairness of pFS is independent to packet size, this section gives the verification using simulation. Without loss of generality, we choose output 0 of an $N \times N$ switch and observe the flows to this port in case of heavy load. Suppose all the flows have the same bandwidth reservation and all the inputs have the same arrival rate $\lambda_i = \sum_{j=0}^{N-1} \lambda_{i,j} = \lambda$. The arrival rate of $F_{i,j}$ is set to be

$$\lambda_{i,j} = \begin{cases} \frac{2\lambda}{N} & j = 0 \\ \frac{\lambda - \lambda_{i,0}}{N-1} & j \neq 0 \end{cases}, \quad i, j = 0, \ldots, N-1, \tag{17}$$

which gives output 0 a heavy load of 2λ. Each flow adopts $T_B(L)$ as the traffic scenario and the maximum packet size L is randomly generated in [1, 24]. (The maximum transmission unit (MTU) of Ethernet is 1500 bytes, which can be segmented into 24 cells.)

Figure 5 highlights the significance of fairness for packet-mode scheduling using an 8×8 switch with $\lambda = 0.9$, the maximum packet size of each flow is given in Fig. 6. It can be seen that:

1. Without fairness mechanism, the bandwidth allocation of packet-mode scheduling is tightly related to the packet size of each flow. Simulation of packet-mode iSLIP (iSLIP-pm) shows that flows with large packets get more bandwidth than those with small packets;
2. Fair bandwidth allocation is achieved in pFS regardless of packet size distribution, the fairness factor calculated according to (1) is 0.006. Meanwhile, no packet loss was observed in the flows to the other outputs, which means 100% throughput is achieved in each of them without being affected by the greedy flows to output 0.

Simulations under various conditions were also carried out (i.e., different switch size N, system load, and traffic scenario), and similar results were obtained.

4.3 Proportional Bandwidth Allocation

In the previous section all the flows to output 0 are greedy, this section studies how excess bandwidth is allocated in case some flows cannot use up their reservations.

To show the results more clearly, we use a 4×4 switch where the reservations of the flows to output 0 are set to be $b_{0,0} = 0.4$, $b_{1,0} = 0.3$, $b_{2,0} = 0.2$ and $b_{3,0} = 0.1$. The input load λ is allocated among the flows as

$$\lambda_{i,j} = \begin{cases} \frac{1}{2}\lambda & \text{if } j = 0 \\ \frac{1}{6}\lambda & \text{if } j \neq 0 \end{cases}, \quad i, j = 0, \ldots, 3, \tag{18}$$

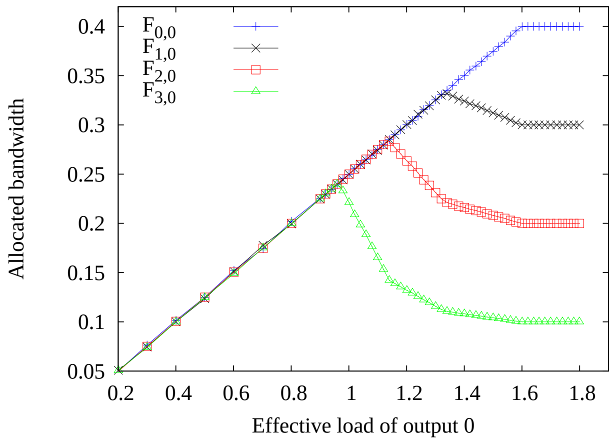

Fig. 7 Bandwidth allocation in output 0.

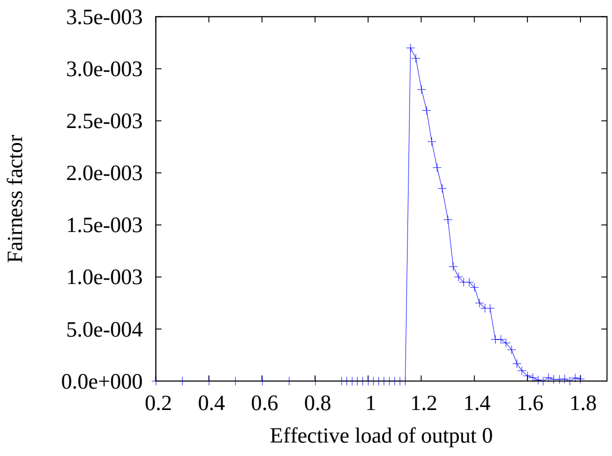

Fig. 8 Fairness of bandwidth allocation in output 0.

which means the effective load of output 0 is 2λ.

Increasing input load λ from 0.1 to 0.9, we get four curves indicating the allocated bandwidth of $F_{0,0}$, $F_{1,0}$, $F_{2,0}$ and $F_{3,0}$, as shown in Fig. 7, from which a curve reflecting the fairness factor is obtained and shown in Fig. 8. The results show:

- $\lambda \leq 1/2$: All the flows conform to their reservations and are fully served.
- $1/2 < \lambda \leq 4/7$: $F_{0,0}$ and $F_{1,0}$ cannot use up their reservations and the excess bandwidth is $B = 0.7 - \lambda$. Note that $F_{2,0}$ has an arrival rate less than $b_{2,0} + \frac{b_{2,0}}{b_{2,0}+b_{3,0}}B$, it is fully served and only $F_{3,0}$ is regulated.
- $4/7 < \lambda \leq 2/3$: Both $F_{2,0}$ and $F_{3,0}$ are regulated, and the ratio of their occupied bandwidth is approximately $2:1$.
- $2/3 < \lambda \leq 4/5$: Three flows are regulated: $F_{1,0}$, $F_{2,0}$ and $F_{3,0}$, and the ratio of their occupied bandwidth is approximately $3:2:1$.
- $\lambda > 4/5$: All the flows exceed their reservations and get just the reserved bandwidth.

Fig. 7 and 8 were obtained by adopting $T_B(L)$ with L randomly generated within [1, 24]. We also simulated other traffic scenarios and found that proportional fairness was well guaranteed.

4.4 Throughput

It has been proved that packet-mode scheduling has no throughput limitation, and iSLIP-pm has been shown to achieve high throughput under admissible traffic[1]. Although pFS employs bandwidth-sensitive match pattern calculation, the throughput sacrifice is observed to be trivial, which can be further reduced to negligible with multiple iterations.

Definition 3: At each slot, suppose there are n inputs requesting for transmission, and the number of match pairs after scheduling is m, the match percentage (MP) of this scheduling is defined to be

$$MP = \frac{m}{n}. \tag{19}$$

Since MP directly affects the throughput performance, we compare pFS with iSLIP-pm to investigate their MP and throughput. To clearly reflect the effect of packet size on throughput of packet-mode scheduling algorithms, the simulation adopts fixed-size packet streams $T_A(L)$. The switch size is set to 16×16 with port bandwidth 1, and all the flows have the same load and bandwidth reservation.

First, the packet size is fixed to be 1, 4 and 8 cells, respectively, and the results of both pFS and iSLIP-pm were obtained with 1 iteration, as shown in Fig. 9 and 10. Under light load, both algorithms achieve 100% throughput. However, pFS experiences slight performance degradation under heavy load.

To further explore the performance of pFS , we fixed the load to 0.99 and carried out simulations by increasing

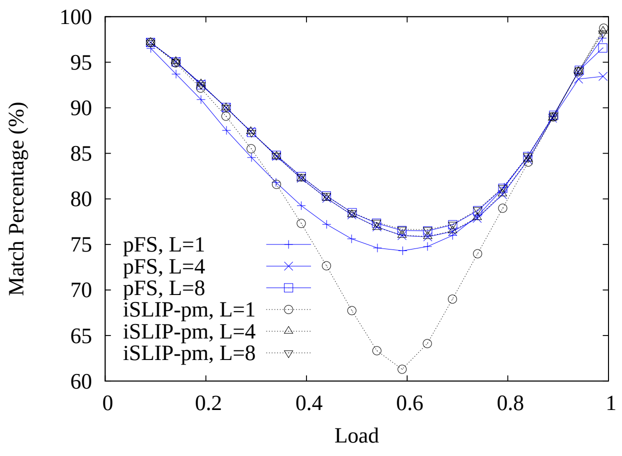

Fig. 9 MP: pFS and iSLIP-pm .

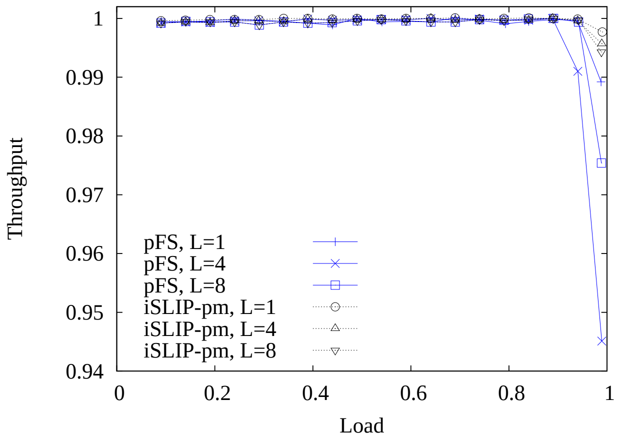

Fig. 10 Throughput: pFS and iSLIP-pm .

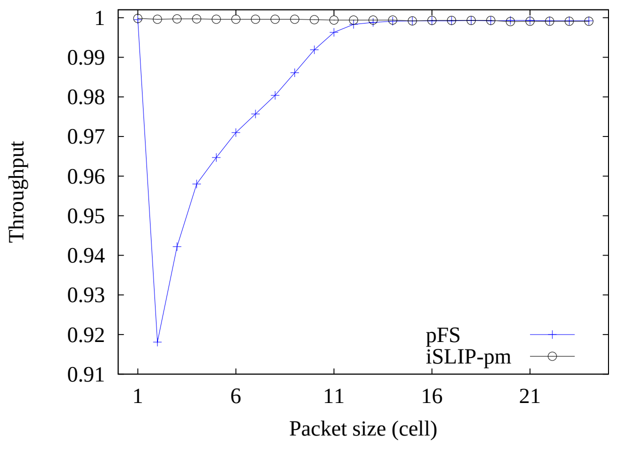

Fig. 11 Throughput: Iteration=1, load=0.99.

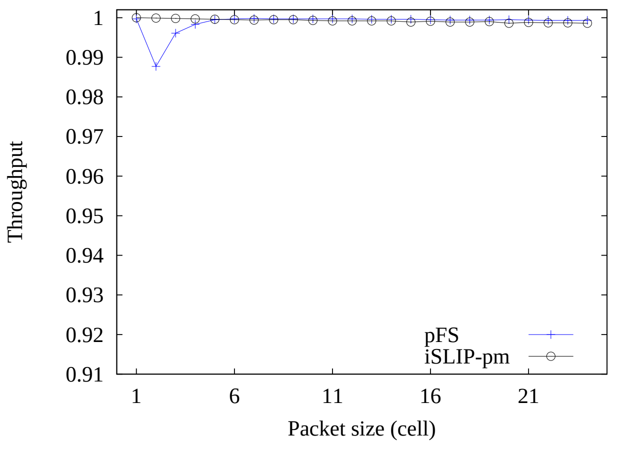

Fig. 12 Throughput: Iteration=2, load=0.99.

the packet size from 1 to 24. The results in Fig. 11 and 12 show that:

1. Although pFS experiences throughput sacrifice under certain traffic scenario, the degradation can be greatly reduced using multiple iterations. For example, 2-iteration improves the worst throughput from 91.8% to 98.9%. Simulation with 4-iteration shows a further increase to 99.9% (which is not included in the figure);
2. The degradation of pFS comes from the credit-based arbiters. One a flow exhausts its credit, it is likely to be skipped temporarily, which brings more port synchronization to pFS compared to iSLIP-pm, e.g., multiple output ports may grant the same input port more frequently. This effect turns to be less serious with large packets and 1-cell packets since the delivering of a large packet leaves more time for the flows to accumulate credit while 1-cell packets seldom turn the credits to negative. In addition, Fig. 12 shows that multi-iteration scheduling is effective to reduce the degradation since slave selectors can be used to match ports with negative credits.

Extensive simulations with *real* traffic pattern T_C were also carried out for various switch sizes, system load, and iterations. The results in Table 1 show that pFS provides satisfying throughput with 2 iterations or more, and the performance with 1-iteration is also acceptable.

4.5 Delay Performance

It has been shown that iSLIP-pm achieves good delay performance [1], so we compare the mean packet delay

Table 1 Throughput of pFS with real traffic.

Load	Iteration	Throughput (%)		
		$N = 8$	$N = 16$	$N = 32$
0.99	1	97.70	96.44	95.92
	2	99.79	99.69	99.40
	4	99.87	99.80	99.79
0.95	1	99.95	99.72	99.20

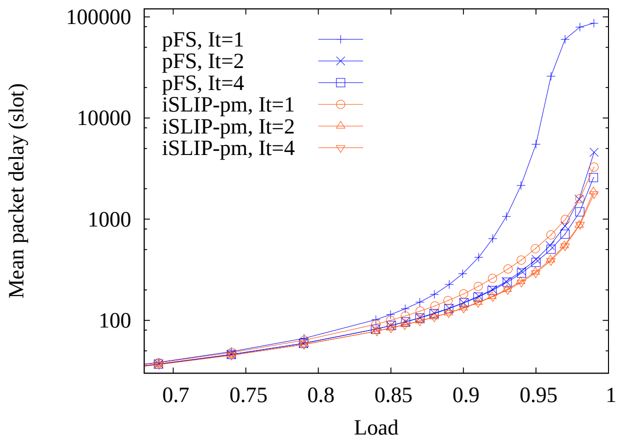

Fig. 13 Delay performance: homogeneous traffic.

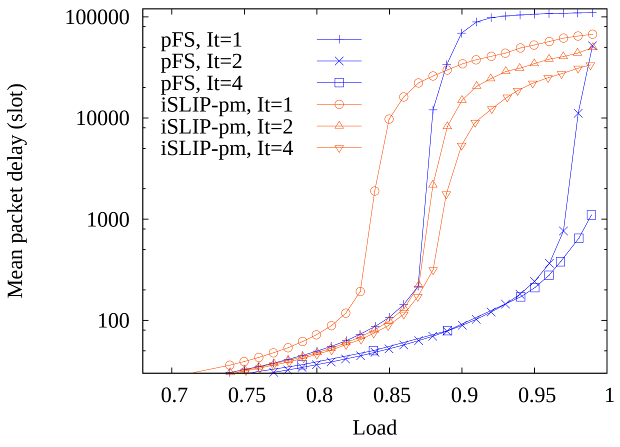

Fig. 14 Delay performance: heterogeneous traffic.

of pFS with iSLIP-pm. The simulation is based on a 16×16 switch where all the flows have the same arrival rate and bandwidth reservation. Two traffic patterns are adopted:

1. Homogeneous Traffic: Each flow has the same packet size distribution and traffic scenario T_C is used;
2. Heterogeneous Traffic: Each flow has different packet size distribution and traffic scenario $T_B(L)$ is used where L is randomly generated within [1, 24]. We believe heterogeneous traffic is of special importance since real applications are diverse and the flows tend to have different statistics during a short period of time.

Figure 13 and 14 are the results under homogeneous and heterogeneous traffic, which show:

1. As discussed in Section 4.4, pFS experiences throughput degradation with 1 iteration, thus its delay performance is not as good as iSLIP-pm under heavy load;
2. Under homogeneous traffic, pFS achieves similar delay performance as iSLIP-pm with multiple iteration;
3. Under heterogeneous traffic, the fair mechanism of pFS makes it remarkably outperform iSLIP-pm with multiple iterations, which is significant for real application where heterogeneous traffic exists with high probability.

4.6 Convergence

Various simulations show that $\log_2 N$ times of iteration are enough to achieve high performance for an $N \times N$ switch. Although we have not been able to prove the upper bound of iteration for pFS to converge, $\log_2 N$ has been shown to be a good empirical bound, which is also consistent with the findings of most iterative algorithms such as iFS , iSLIP , PIM, etc. Theoretical analysis of the convergence bound is our future work.

5. Hardware Implementation

This section presents the hardware implementation of the key modules: credit-based arbiter and frame counter. It will be shown that the computation of both modules can be completed within a single clock cycle (e.g., 10 ns in case of 100 MHz clock), thus is applicable to high speed switches. The overall architecture of the scheduler can be

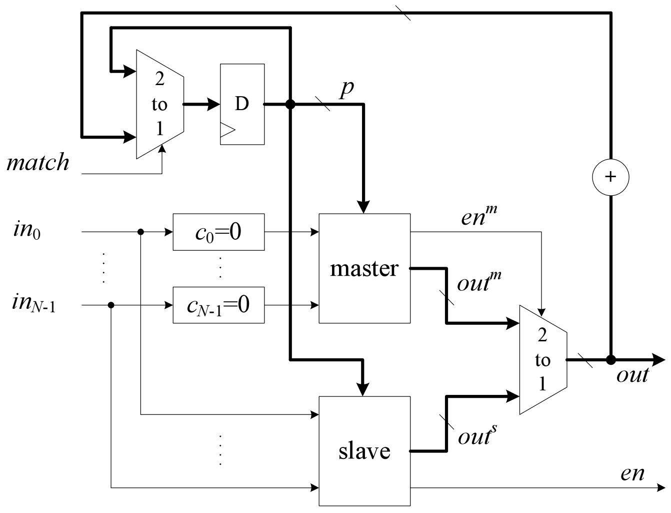

Fig. 15 Block diagram of the credit-based arbiter.

found in [11].

5.1 Credit-based Arbiter

The structure is illustrated in Fig. 15. The master and slave selectors are constructed with priority encoders where the highest priority is programmable according to pointer p. The output of each encoder consists of an index and an enable indicator, where en^m is used to make selection between out^m and out^s, while en^s indicates whether out is effective. Finally, pointer p is updated to $out + 1$ if a match is set up. With this structure, each arbitration can be completed within a single clock cycle.

5.2 Frame Counter

From (5) it can be seen that the most complex computation for the frame counter takes place when FC reaches 1, where multiple additions have to be completed. However, this operation can be distributed to multiple slots to facilitate hardware implementation. Suppose the number of *active* and *waking* flows is M $(M \leq N)$ at the time $FC(n) = 1$, their bandwidth reservations can be denoted with $d'_0, \ldots, d'_{M-1}$. It is shown in (7) that only the time FC reaches 1 is important to the scheduler, thus (5) can be replaced with

$$FC(n+k+1) = \begin{cases} FC(n+k) + d'_k - s(n+k) & k < M \\ FC(n+k) - s(n+k) & k \geq M \text{ and } FC(n+k) \neq 1 \end{cases} . \tag{20}$$

By choosing $d_i \geq 2$ $(i = 0, \ldots, N-1)$ in (6), $FC(n+k+1) > 1$ is always ensured when $k < M$ and the addition is distributed to the first M slots.

Hardware diagram of the counter is shown in Fig. 16. When FC reaches 1, mk_i is set to be h_i from (4) and is registered. The priority encoder selects a non-zero mk_i, say mk_i, and the corresponding d_i is added to FC, then mk_i is cleared and the next one is selected. When all the mk_i are cleared, the output of the priority encoder becomes non-effective and d' turns to be 0, in this case, $FC(n)$ either behaves like a plain decrement counter when $sent(n) = 1$ or keeps unchanged when $sent(n) = 0$. Using this structure, the computation of FC in each slot can be completed within a single clock cycle.

6. Conclusion

This paper presents a packet-mode fair scheduling algorithm for input-queued switches. Different from existing

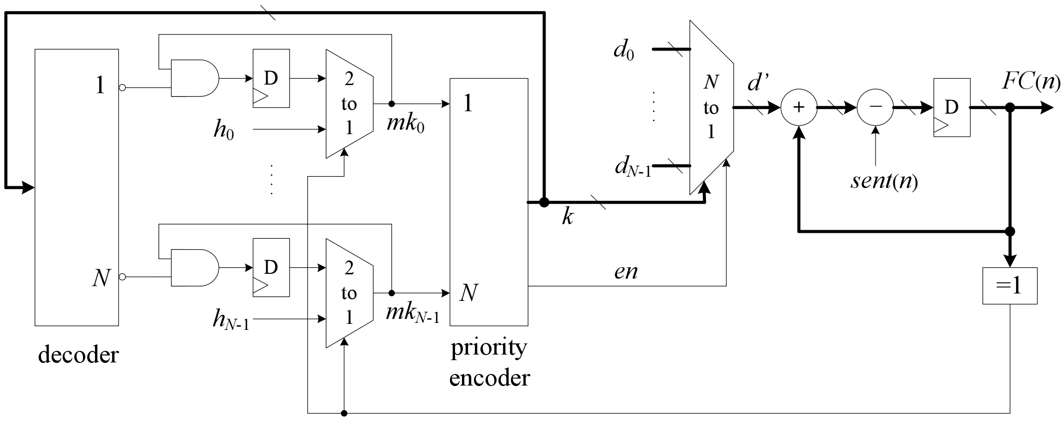

Fig. 16 Block diagram of the frame counter.

algorithms, pFS achieves proportional fairness under the context of packet-mode scheduling, where all the segments of a packet are delivered from the input to the output port contiguously to facilitate reassembly and improve performance.

The fairness of the proposed algorithm is proportional: the reserved bandwidth of each flow is guaranteed, and the excess bandwidth is allocated among the competing flows proportionally to their reservation. pFS benefits QoS-aware networks by allocating bandwidth according to the reservation, at the same time, it is also effective to achieve uniform bandwidth assignment in best-effort networks so as to prevent misbehaving flows from suppressing normal ones. The fairness of pFS is independent to traffic patterns.

While providing fairness guarantee, pFS also achieves high throughput and low delay. Compared to packet-mode iSLIP, pFS is able to provide identical delay under homogeneous traffic and achieves remarkable performance improvement under heterogeneous traffic.

The hardware implementation of pFS is presented, where the algorithm is shown to have low complexity and the fairness-related computation can be completed within a single clock cycle, which is highly favorable to high speed switches.

References

[1] M.A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Packet-mode scheduling in input-queued cell-based switches," IEEE/ACM Trans. Networking, vol.10, no.5, pp.666–678, Oct. 2002.

[2] V. Tabatabaee and L. Tassiulas, "MNCM a new class of efficient scheduling algorithms for input-buffered switches with no speedup," Proc. IEEE INFOCOM'03, pp.1406–1413, April 2003.

[3] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz, "Tiny Tera: A packet switch core," IEEE Micro, vol.17, no.1, pp.27–40, Feb. 1997.

[4] Cisco, "Cisco 12000 Gigabit switch router." [Online] http://www.cisco.com.

[5] Lucent, "GRF-MultiGigabit routers, product overview." [Online] http://www.lucent.com.

[6] M. Rosenblum, M. Goemans, and V. Tarokh, "Universal bounds on buffer size for packetizing fluid policies in input queued, crossbar switches," Proc. IEEE INFOCOM'04, March 2004.

[7] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," J. of Internetworking Research and Experience, vol.1, no.1, pp.3–26, June 1990.

[8] N. Ni and L.N. Bhuyan, "Fair scheduling and buffer management in internet routers," Proc. IEEE INFOCOM'02, pp.1141–1150, June 2002.

[9] S. Kumar and A. Kumar, "On implementation of scheduling algorithms in high speed input queuing cell switches," Proc. IEEE ICC'03, pp.152–157, May 2003.

[10] L. Mhamdi and M. Hamdi, "Practical scheduling algorithms for high-performance packet switches," Proc. IEEE ICC'03, pp.1659–1663, May 2003.

[11] N. Mckeown, "The $i$slip scheduling algorithm for input-queued switches," IEEE/ACM Trans. Networking, vol.7, no.2, pp.188–201, April 1999.

[12] N. Mckeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," IEEE Trans. Commun., vol.47, no.8, pp.1260–1267, Aug. 1999.

[13] N. Mckeown, Scheduling algorithm for input-queued cell switches, Ph.D. thesis, University of California, Berkeley, 1995.

[14] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High speed switch scheduling for local area networks," ACM Transactions on Computer Systems, vol.11, no.4, pp.139–352, Nov. 1993.

[15] H.J. Chao, "Saturn: a Terabit packet switch using dual round-robin," IEEE Commun. Mag., vol.38, no.13, pp.78–84, Dec. 2000.

[16] Y. Li, S. Panwar, and H.J. Chao, "On the performance of a dual round-robin switch," Proc. IEEE INFOCOM'01, pp.1688–1697, April 2001.

[17] G. Nong and M. Hamdi, "Burst-based scheduling algorithms for non-blocking atm switches with multiple input queues," IEEE Commun. Lett., vol.4, no.6, pp.202–204, June 2000.

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

[19] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round-robin," IEEE/ACM Trans. Networking, vol.4, no.3, pp.375–385, June 1996.

[20] S.S. Kanhere, H. Sethu, and A.B. Parekh, "Fair and efficient packet scheduling using elastic round-robin," IEEE Trans. Parallel Distrib. Syst., vol.13, no.3, pp.324–336, March 2002.

[21] D. Stiliadis, Traffic scheduling in packet-switched networks: analysis, design, and implementation, Ph.D. thesis, University of California, Santa Cruz, June 1996.

[22] M. Andrews and M. Vojnovic, "Scheduling reserved traffic in input-queued switches: new delay bounds via probabilistic techniques," Proc. IEEE INFOCOM'03, pp.764–774, April 2003.

[23] X. Zhang and L. Bhuyan, "Deficit round-robin scheduling for input-queued switches," IEEE J. Select. Areas Commun., vol.21, no.4, pp.584–594, May 2003.

[24] K. Thompson, G.J. Miller, and R. Wilder, "Wide-area internet traffic patterns and characteristics," IEEE Network, vol.11, no.6, pp.10–23, Nov. 1997.