Fair Scheduling in Internet Routers

Nan Ni, *Member*, *IEEE*, and Laxmi Narayan Bhuyan, *Fellow*, *IEEE*

Abstract—Input buffered switch architecture has become attractive for implementing high performance routers and expanding use of the Internet sees an increasing need for quality of service. It is challenging to provide a scheduling technique that is both highly efficient and fair in resource allocation. In this paper, we first introduce an iterative fair scheduling(*i* FS) scheme for input buffered switches that supports fair bandwidth distribution among the flows and achieves asymptotically 100 percent throughput. The *i*FS is evaluated both under synthetic workload and with Web traces from the Internet. Compared to the commonly used synthetic input, our simulation results reveal significant difference in performance when the real network traffic is employed. We then consider fair scheduling under various buffer management mechanisms and analyze their impact on the fairness in bandwidth allocation. Our studies indicate that early packet discard in anticipation of congestion is necessary and per-flow based buffering is effective for protecting benign users from being adversely affected by misbehaved traffic. Buffer allocation according to bandwidth reservation is especially helpful when the input traffic is highly bursty.

Index Terms—Quality of service, fair bandwidth allocation, switch scheduling, buffer management, decongestion mechanism, web traffic.

1 Introduction

The exponential growth of the Internet has put increasing demands on the routers and switches in the network for high bandwidth and low latency. In addition, as networks provide new services supporting multicast, voice, security, and bandwidth reservation, quality of service (QoS) is becoming a major issue in the design of routers [25], [18]. Fairness in resource allocation is very important to support the need for diverse applications. Fair queuing algorithms have been developed [8], [38] to schedule packets at an output link of a router. However, little research has been done to address QoS issues inside the router operation itself. In fact, this is crucial because the router is the one responsible for forwarding packets down the stream. Without the router taking action, fair queuing at output is left to the mercy of traffic flow from upstream and the resource may not be distributed as fairly as desired. The purpose of this paper is to develop fair scheduling and buffer management schemes for Internet routers and to demonstrate their superiority using actual Web traces from NLANR [27] and UCB [37].

A router consists of three parts, namely, 1) line cards that connect to datalinks, 2) a router processor that runs routing protocols, and 3) a backplane crossbar switch that actually transfers the packets or cells from inputs to outputs. In this paper, we are mainly concerned with QoS issues of the backplane switch design in a router. The switch has buffers either at the input or output to store the packets temporarily during transmission. Although output buffering can achieve better throughput, it is known to suffer from poor scalability [17]. The reason is that the output port of an $N \times N$ switch has to operate N times faster than the input in order to accommodate requests on all possible inputs. Consequently, most high-performance routers employ input queues with their crossbar backplanes [24], [30]. Also noticable is that these crossbars operate in a cyclic fashion, arbitrating and transfering fixed sized packets (also called cells) in each cycle. The cell-based switch is chosen in the high performance router for its potentially high throughput. On the QoS side, unlike switches with variable length packets where a large packet from an ill-willed source could dominate for a substantial period of time, the cell-based switch is suitable for fine tuning for fairness. For these reasons, we will assume a cell-based switch unless otherwise stated.

In this research, we develop fair scheduling schemes for input-buffered switches. High throughput and fairness in resource allocation are contradictory goals in a switch design. Due to the fact that only a single cell can be transmitted from each input of the crossbar in a given slot, cells forwarded based on a maximal set of input-output match may not coincide with those satisfying fairness. On the other hand, passing cells based on fair resource allocation may not produce the highest throughput and may give rise to underutilization of the crossbar switch. The aim of this research is to find scheduling and buffer management techniques that are both fair and highly efficient for link utilization.

According to [8], bandwidth allocation of a link is fair if, for each flow, the received bandwidth is proportional to its share of reservation. We know that the fluid flow queuing (or Generalized Processor Sharing (GPS) [29]) algorithm, which sends packets in a bit-by-bit round-robin fashion, is absolutely fair but unrealistic. Numerous fair queuing algorithms have been developed to approximate the GPS algorithm in units of packets (see [38] for a survey). However, most work on fair queuing has been conducted in the context of output queuing due to its conceptual

- *N. Ni is with IBM Corp., 11400 Burnet Rd., Austin, TX 78758. E-mail: ni_nan@us.ibm.com.*
- *L.N. Bhuyan is with the Department of Computer Science and Engineering, University of California, Riverside, CA 92521. E-mail: bhuyan@cs.ucr.edu.*

Manuscript received 24 Jan. 2001; revised 17 Dec. 2001; accepted 24 Jan. 2002.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 115752.

simplicity. The situation becomes complicated in the case of an input buffered switch, where flows not only contend for output link bandwidth but also have to compete to access the crossbar. Nevertheless, in order to achieve high performance as well as to provide quality of service, it is imperative to maintain fairness in input buffered switch scheduling.

In the first part of this paper, we propose an iterative fair scheduling (*i*FS) scheme which provides high throughput, low latency, as well as fair bandwidth distribution among the contesting flows. As with other iterative approaches, *i*FS tries to increase the number of input-output matches during each iteration, but the major difference is that, instead of picking a cell from each input in a probabilistic ([1], [13]) or round-robin ([22], [34]) fashion, a cell is chosen based on the bandwidth requirement of the flows. In this way, we are able to allocate bandwidth to various flows in proportion to their reservation and prevent misbehaving flows from taking advantage at the expense of others. The *i*FS is compared with *i*SLIP [22] and WPIM [36] for throughput and fairness.

In a real deployment of a network with finite buffering space, packets are dropped in the presence of congestion. In order to support fair bandwidth allocation, it is crucial to determine when to drop and which packet to drop. Thus, the fairness in scheduling issue should be considered in conjunction with the decongestion mechanism. In this paper, four decongestion mechanisms are studied and their impact on the fair sharing of bandwidth is examined.

The effectiveness of our fair scheduling approaches is demonstrated by simulations using a cycle-based simulator. Experiments are conducted with different types of input workload. Investigation has shown that Internet traffic is very bursty in nature [31], [7] and, therefore, cannot be characterized very well by the commonly employed Bernoulli or geometrically distributed on/off traffic model. While distributional models have been developed for the Internet workload [4], traffic traces are widely applied for evaluating web servers [2], proxy caches [6], [21], and packet forwarding methods [16]. The use of real traffic traces can offer direct validation of the network components under study. Hence, we incorporate the traffic pattern directly from the Internet, in addition to employing synthetic workload, and we are not aware of any existing application of traces in the context of router design or switch scheduling.

To summarize, we have the following original contributions in the paper:

- We propose an iterative fair scheduling (*i*FS) scheme for unicast traffic. It supports fair bandwidth allocation and achieves asymptotically 100 percent throughput with uniform traffic.
- We analyze various buffer allocation policies, along with fair scheduling, and pinpoint their effect on fair bandwidth allocation.
- Ours is the first attempt to evaluate switch scheduling schemes using real traffic from the Internet. It gives some insight into the difference in performance compared to the common practice of evaluating through synthetic workload.

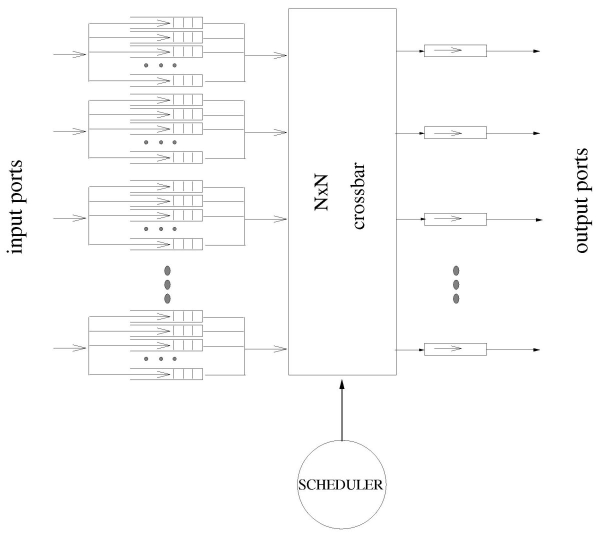


Fig. 1. Block diagram for an input buffered switch architecture.

The rest of the paper is organized as follows: Section 2 gives an overview of the related work on scheduling for input buffered switches. In Section 3, we propose the iterative fair scheduling (*i*FS) for unicast traffic. Extensive simulation results with synthetic workload, as well as traces from the Internet, are presented in Section 4. Buffer management schemes are examined in Section 5. Finally, Section 6 concludes the paper.

2 Related Work

Input queuing is subject to the head of line (HOL) problem with maximum throughput of 58.6 percent using the FIFO input queue [17]. A solution has been found which constructs separate queues at each input so that a packet will not be blocked by packets going to other destinations. Major issues in the input buffered switch architecture design include buffer management and scheduling. We have studied various queuing and buffer allocation schemes in our previous research [9], [33]. In this paper, we seek to address the issue of fairness in scheduling (i.e., input to output matching) for quality of service. Fig. 1 gives a block diagram of the cell-based switch architecture we consider throughout the paper. It consists of four components: input ports, output ports, crossbar switching fabric, and a scheduler. At any given switch cycle (slot), at most one cell from each input can be routed to the output side and each output can only accept at most one cell. As a result, the scheduling is more intricate than that for the output queued switch. Most existing work on scheduling for input buffered switch attempts to achieve high throughput by looking for maximum bipartite matching between inputs and outputs. Schemes such as PIM [1], *i*SLIP [22], and Shakeup [13] repeatedly search for matches at each time of scheduling. Some approaches can achieve 100 percent throughput asymptotically [23].

An iterative scheduling algorithm essentially consists of three major steps in each iteration, as illustrated in Fig. 2.

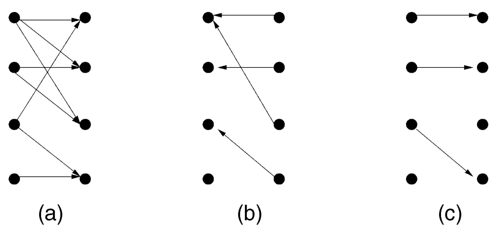


Fig. 2. Three-step procedure in iterative matching.

1. *Request stage:* An input may have several requests. Each unmatched input sends requests to the outputs for which it has cells for.
2. *Grant stage:* There may be several requests to an output. Each unmatched output chooses one from several received requests and sends a grant signal to one of the inputs.
3. *Accept stage:* Each unmatched input may receive grant signals from several outputs. Upon receiving grant signals, each input sends an accept signal to only one of the outputs offering the grants.

Probabilistic Iterative Matching (PIM), suggested by Anderson et al. [1], is the first switch scheduling algorithm that employs an iterative approach. At the grant stage, each output sends out a grant signal to a randomly selected requesting input. Again, at the accept stage, each input accepts one output from several grant signals at random. It takes an average of $O(logN)$ iterations for the PIM to converge. The consequence of random selection in the grant and accept stage of PIM is that an input could possibly end up not being served for a long time. Thus, this scheme is not starvation free. Moreover, implementation of random selection among the members of a time-varying set is not an easy task.

The *i*SLIP scheme proposed by McKeown [22] uses rotating priority (round-robin) arbitration to schedule active inputs and outputs in turn. A grant pointer is kept for each output to track the input with the highest priority. Similarly, there is an accept pointer for each input which tells the output with the highest priority. Whenever a match is found, the corresponding grant/accept pointers are incremented (modulo the number of ports). Compared to PIM, *i*SLIP is simpler in implementation [15] and achieves higher throughput [22].

Matches are irrevocable in both PIM and *i*SLIP, which means that later iterations can only add upon previously made matches, but cannot change them even if better matches can be found. To escape this "local maximum", Goudreau et al. offer a Shakeup technique in [13] where each unmatched input is allowed to force a match for itself randomly even though an existing match has to be knocked off. In other words, Shakeup attempts to find the global maximum, but the feasibility of its implementation in a real system is unknown because it takes more iterations to converge.

The issue of fair resource allocation is not considered in the above-mentioned approaches. Iterative schemes that do address the fairness issue include statistical matching [1] and Weighted Probabilistic Iterative Matching (WPIM) [36]. Statistical matching is similar to PIM except that the matching process is now initiated by the outputs, each generating a grant signal to a randomly selected input based on the reserved proportion of bandwidth. It can happen that an input is picked when its queue is empty and the switch is poorly utilized. WPIM is also built upon PIM, where, based on the reservation, every input flow is assigned a quota that can be used in a frame of a constant number of slots. During each frame, flows that have not reached their quotas secure an equal share of bandwidth by random selection, as is done in PIM. To accomplish this, an additional masking stage is added to the 3-step procedure to exclude those inputs that have consumed their quotas in the current frame.

Scheduling fairness is also addressed in other work under assumptions not relevant to this paper. Stephens and Zhang [35] considered a switch that has an effectively fully connected crossbar [33] and processes packets of variable length. There, the inputs and outputs are decoupled so that each output independently reads from any input with a packet for it. Li and Ansari gave an end-to-end delay bound provided that incoming flows conform to the (r, T) traffic model [19]. Recent efforts in designing scheduling algorithms capable of providing QoS for input queued switches is presented by Nong and Hamdi in [28].

Among the packet discarding schemes, some assume a queuing discipline of FCFS, others are designed with per-flow queuing. The most basic one among the FCFS disciplines is the "do nothing" policy, which is a complete buffer sharing with a *drop-tail* (DT) mechanism, where cells are dropped when they arrive to find the buffer full. Generally, network systems do not support cell level retransmission, so a partially received packet is of no value. In *partial packet discard* (PPD) [3], after a cell from a packet is dropped, all subsequent cells of the same packet are dropped as well. In *early packet discard* (EPD) [11], the entire packet (i.e., all the cells constituting a new packet) is dropped whenever congestion is anticipated because the buffer occupancy exceeds a certain threshold. A more sophisticated policy by Floyd and Jacobson is *random early discard* (RED) [10], whose primary goal is to avoid performance degradation and unfairness caused by DT. It does so by maintaining average buffer occupancy at a level significantly below the total number of buffers. To achieve this, packets are dropped with a certain probability when the average buffer occupancy reaches a certain level. Drop probability increases with the average queue occupancy and, once the queue occupancy exceeds a maximum buffer threshold, all arriving packets are discarded. All these schemes assume an FCFS scheduler, each arriving packet is treated identically, and all the flows see the same loss rate.

Examples of per-flow discarding include *longest queue drop* (LQD) [12] and *fair buffer allocation* (FBA) [26]. The justification behind LQD is that if flows are given equal weights, the ones that use the link more tend to have longer queues. Hence, biasing packet discarding such that flows with longer queues have a higher drop rate should make the bandwidth sharing more fair. In FBA, buffer share for each flow is computed as a function of the number of free buffers and the number of active connections. Once the aggregate occupancy is above the specified threshold, packets arriving for flows that have occupied more than the fair shares are thrown away.

3 Iterative Fair Scheduling Scheme

In this section, we first introduce a definition of fairness in input buffered switch scheduling and then propose an iterative fair scheduling (*i*FS) scheme that can be used to achieve fair bandwidth allocation.

Let $f(i,j)$ denote the *j*th flow from input i. It goes to output $d_{i,j}$ and reserves a bandwidth of $B_{i,j}$. The number of flows from input i is n_i. Let $t_{i,j}(t_1,t_2]$ be the amount of traffic (in bits) reaching the output from flow $f(i,j)$ in time interval $(t_1,t_2]$. We say that two flows $f(i_1,j_1)$ and $f(i_2,j_2)$ are in contention if $i_1 = i_2$ or $d_{i_1,j_1} = d_{i_2,j_2}$. Note that we do not define precisely what a *flow* is because we would like our *i*FS scheme to be applied in a broad sense: Depending on the desired granularity of QoS and scalability concern, a flow can be as fine as a TCP connection or as coarse as a class of aggregated connections with a similar bandwidth requirement.

Following the definition of the GPS server for a single shared resource in [29], we consider a scheduling scheme for input buffered switches to be fair as follows:

Definition 1. *For any two back logged flows $f(i_1,j_1)$ and $f(i_2,j_2)$ that are in contention, a scheduling scheme is fair in $(t_1,t_2]$ if*

$$\frac{t_{i_1,j_1}(t_1,t_2]}{t_{i_2,j_2}(t_1,t_2]} = \frac{B_{i_1,j_1}}{B_{i_2,j_2}}.$$

This definition specifies the ideal situation, but if the output link bandwidth is to be best utilized (i.e., work conserving), it is possible that the equation may not be held under all the combinations of bandwidth reservations. The discrepancy of this definition from the one of GPS [29] is that, here, the flows are contesting to access the crossbar as well as to access the output links. In other words, we are trying to allocate correlated resources because a cell must first make its way out of the input buffer before going to the intended output.

Nevertheless, we find that the existing fair queuing algorithms, which have been successful in allocating a single shared resource, can be applied here to facilitate our effort to support fair bandwidth distribution. In a fair queuing algorithm, a virtual system time $V(t)$ (corresponding to the number of rounds made till time t in GPS) is maintained. Every incoming packet is assigned a virtual starting time and a virtual finishing time, depending on its bandwidth requirement. Virtual starting time and virtual finishing time denote the virtual time when a packet should begin and finish sending if GPS were used. The transmitting order is then regulated according to nondecreasing starting time [14], nondecreasing finishing time [8], or a combination of both [5]. Naturally, flows with larger bandwidth reservations will take a greater proportion of bandwidth since they tend to have a smaller virtual starting time (or finishing time).

Our basic idea of *i*FS is to enhance the iterative approaches described in Section 2 by giving out grant signals from the individual outputs based on virtual time and then trying to resolve input contention in the accept stage.

For each output link, we maintain a fair queuing engine, which assigns a virtual time to every incoming cell based on bandwidth reservation of the flow. Note that the arriving cells are queued in the input buffer first-in-first-out on a per flow basis. This is required for implementing a fair queuing algorithm where each output must keep track of the active flows to compute the virtual time. It is different from other iterative approaches where input queues are arranged on a per output basis. Also note that, by applying the fair queuing algorithm, each output has the knowledge of the active flows destined to it, so we can initiate the scheduling from the outputs and save the request stage in the common 3-step procedure (Section 2).

In each iteration, the first step in our scheme is for every unmatched output to independently send a grant signal to one of the unmatched inputs which has the cell with the minimal virtual time corresponding to that output. Such a cell is marked as a "candidate" with respect to its input. It may so happen that an input receives multiple grants from different outputs and has multiple candidates, which means that several flows from this input have a minimal value of virtual time with their relevant outputs. Then, each unmatched input selects among its candidates a cell (called "winner"), with the oldest age at the switch, and sends an accept signal to its desired output. That is, in the accept stage, an input resolves the contention on a first-come-first-served (FCFS) basis. The justification is that, since these candidate cells are from the same upstream node, the fact that the cell arrives first among the contending cells implies that it has the smallest virtual time among them at the previous node and, naturally, should be the first one to depart from the current node.

In summary, the *i*FS scheme can be formalized as the following:

- Initially, all inputs and outputs are considered as unmatched and none of the inputs have any candidates.
- Then, in each iteration:
    1. *Grant stage:* Each unmatched output selects a flow with the smallest virtual time for its head-of-line cell and marks the cell as a candidate for the corresponding input. A grant signal is then given to the input.
    2. *Accept stage:* Each unmatched input examines its candidate set, selects a winner according to age, and sends an accept signal to its output. The input and output are then considered as matched. Reset the candidate set to empty.
- At the end of each switch cycle, the winning cells are transferred from the input side to the output side.

Notice that, although a fair queuing algorithm is applied in *i*FS, the access order to an output ruled by the fair queuing algorithm is not strictly followed due to performance concerns. Consider the example in Fig. 3, where flows $f(1,1)$, $f(1,2)$, and $f(2,1)$ are going to output 1, 2, and 1, respectively. Suppose that, at a certain time t, the head-of-line

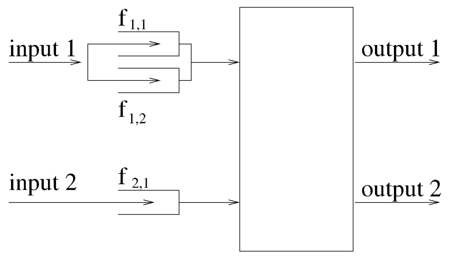


Fig. 3. Fair queuing ordering may not be strictly followed in *i*FS.

cell from $f(2,1)$ has greater virtual time than that from $f(1,1)$. In the first iteration, output 1 gives grant to flow $f(1,1)$ and output 2 gives grant to $f(1,2)$ according to the fair queuing algorithm. Also assume that the head-of-line cell of $f(1,2)$ has older age than that of $f(1,1)$. Hence, input 1 sends an accept signal to output 2 and both input 1 and output 2 are marked as matched. In the next iteration, output 1 finds flow $f(2,1)$ with the smallest virtual time because $f(1,1)$ from input 1 has been excluded. As a result, input 2 and output 1 are matched and we see that the head-of-line cell from flow $f(2,1)$ is transmitted before that of flow $f(1,1)$, even though the latter has a smaller virtual time. This "out-of-order" transfer actually improves the throughput because, otherwise, output 1 would be left idle in the current cycle. Fortunately, this violation only has an instant effect and will not yield unfairness. Continue with the example. In the next time of scheduling, the cell from $f(1,1)$ still has the smallest virtual time for output 1. Flow $f(2,1)$ gets "penalized" since the virtual time for its head-of-line cell at this point is even greater.

Compared to other iterative scheduling schemes, we see that *i*FS uses two steps in each iteration, whereas *i*SLIP and WPIM use three and four steps, respectively. Grants in *i*SLIP are given in round-robin manner without respect to bandwidth reservation. WPIM guarantees fair bandwidth sharing, but in a much coarser granularity. Before running out of quota in WPIM, every flow has equal access to the bandwidth. Consequently, the flows with larger reservations get more bandwidth only after other flows run out of their quotas, which usually happens toward the end of a frame. If we look at the bandwidth distribution within a frame, the bandwidth share is disproportional. Statistical matching also has only two stages. However, since grants are given by outputs to inputs randomly, it is possible that an input port is selected to receive a grant signal when its queue is empty. It has been shown in [36] that statistical matching has inferior performance to WPIM. Unlike Shakeup where matches in prior iterations can be modified, matched input-output pairs are ruled out in later iterations in *i*FS. Although Shakeup is attractive theoretically, its feasibility for high performance switch is unclear because one "knock-off" of an existing match could possibly trigger a chain of "knock-offs" which would take more iterations to converge to global maximum matches.

The issue we do not consider in detail here is the implementation feasibility. Efficient implementation of N fair queuing engines is crucial to the *i*FS for an $N \times N$ switch. It is to our advantage that the switch under discussion is cell-based. The calculation of virtual time can be simplified due to fixed packet length and the fair queuing engine is boiled down to an engine for weighted round-robin instead. In comparison, the switch proposed in [35] needs $3N$ fair queuing engines to deal with variable length packets for an $N \times N$ switch. A second advantage is that, unlike PIM, WPIM, or Shakeup, no random number generator is needed for *i*FS, which again greatly reduces the implementation complexity.

4 Performance Evaluation of *i*FS

Following the criteria in [32] for assessing a resource allocation scheme, we evaluate the proposed *i*FS in two aspects: efficiency and fairness. The principal metrics for efficiency are throughput and delay. A cell-based simulator is developed and the simulations are conducted with the assumption of infinite buffer size. The offered load refers to the probability that a cell arrives at an input in a given slot. Our methodology of evaluation is as follows: We start by examining *i*FS and other approaches using synthetic workload under cases where destinations are uniformly and nonuniformly distributed among the outputs. Results are presented to show that *i*FS can achieve average cell latency and overall throughput close to the existing schemes. Then, we compare the ability of various approaches to support fair bandwidth distribution. In the second part of this section, traces from Internet traffic are applied to the simulator to study how *i*FS works in the "real world." Our results demonstrate that, under both synthetic and real network traffic, *i*FS can achieve high throughput, like *i*SLIP, and, at the same time, support fair bandwidth allocation.

4.1 Measurement from Synthetic Workload

We begin by evaluating the throughput and the average delay of *i*FS under benign i.i.d. Bernoulli traffic, where, at any given slot, a cell arrives with the probability determined by the offered workload. The simulation is performed on a 16×16 switch with 16 flows per input, each destined for a different output, for a total of 256 flows. In the first scenario, each input port sends cells with destinations uniformly distributed among all the output ports. Fig. 4 shows the delay versus the offered load for *i*FS and *i*SLIP with the number of iterations equal to four. The *i*SLIP is known to offer the lowest delay for an input buffered switch. The delay using the output buffered switch is also shown for comparison. Under this circumstance, we observe that the average cell delay for *i*FS is almost identical to *i*SLIP and both are very close to the output buffered switch, which is the lower bound for the delay. Hence, *i*FS has the potential of achieving asymptotically 100 percent throughput for uniform traffic.

Next, we consider a nonuniform Bernoulli traffic model, as used in [36]. The assumption is that four of the switch ports are connected to servers and the remaining 12 to clients. Each client sends 10 percent of its generated traffic to each of the four servers and the remainder is uniformly distributed among the other clients. Similarly, each server directs 95 percent of its traffic to the clients and the remaining 5 percent to the other servers. Fig. 5 indicates that *i*FS is very close to *i*SLIP and WPIM in terms of average cell latency and can reach a throughput of 78.5 percent.

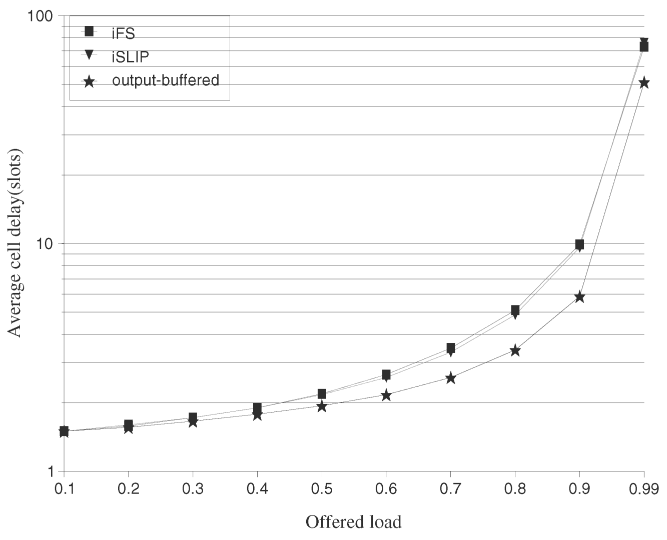


Fig. 4. Performance of *i*FS and *i*SLIP under uniform traffic.

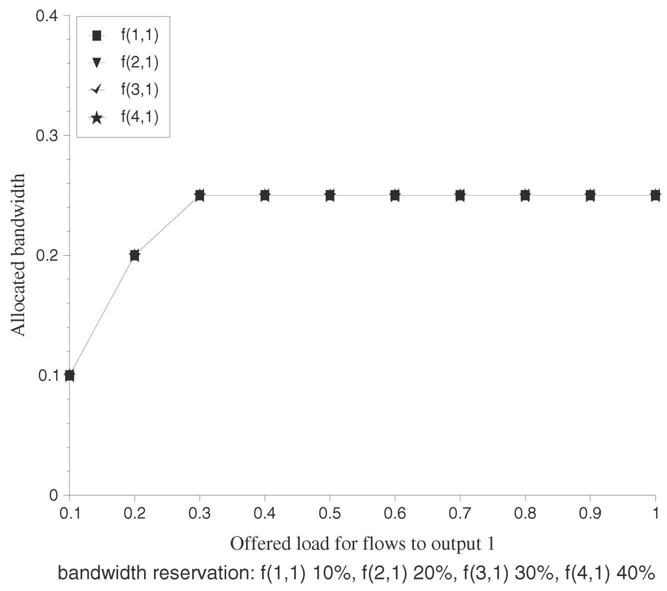


Fig. 6. Received bandwidth using *i*SLIP.

Now, we turn to examining the effectiveness of *i*FS to support the fair bandwidth sharing when a link is overloaded. This time, we simulate a 4×4 switch so that we can plot the results with clarity. Assume that every input has four flows, each going to a different output. Without loss of generality, let the *j*th flow from link i go to output j and denote it as $f(i, j)$, following the notation in Section 3. Also assume that the flows to output 1, $f(1,1)$, $f(2,1)$, $f(3,1)$, and $f(4,1)$, have reserved 10 percent, 20 percent, 30 percent, and 40 percent of the bandwidth, respectively. But, they always maintain the same actual arrival rate. Others are background flows with a load of 5 percent each. We vary the input rate of the flows to output 1 and plot the received bandwidth share in Fig. 6 and Fig. 7 using *i*SLIP and *i*FS scheduling. For a workload under 25 percent, the throughput for every flow is able to keep up with the input workload for both the schemes. However, for a workload beyond 25 percent, all four flows are still treated equally in *i*SLIP and, therefore, they obtain the same share of bandwidth (each 25 percent) despite the variance in bandwidth reservation. The *i*FS, on the other hand, is observed to differentiate the flows according to the promised share when the load is greater than 25 percent. Ill-behaved flows are prevented from influencing the well-behaved ones. For a load beyond 40 percent, each flow receives its allocated bandwidth. Let us look closely at the sharing when input load is between the range of 25 percent and 40 percent. At 30 percent, for instance, flow $f(4,1)$ does not consume its share of 40 percent of bandwidth. The unused part is distributed to flow $f(1,1)$ and flow $f(2,1)$ so that they acquire a larger fraction of bandwidth than their reservations. Flow $f(1,1)$ receives 13.3 percent (versus a reservation of 10 percent) and flow $f(2,1)$ receives 26.6 percent (versus a reservation of 20 percent). Such behavior also conforms to the fairness requirement in [8],

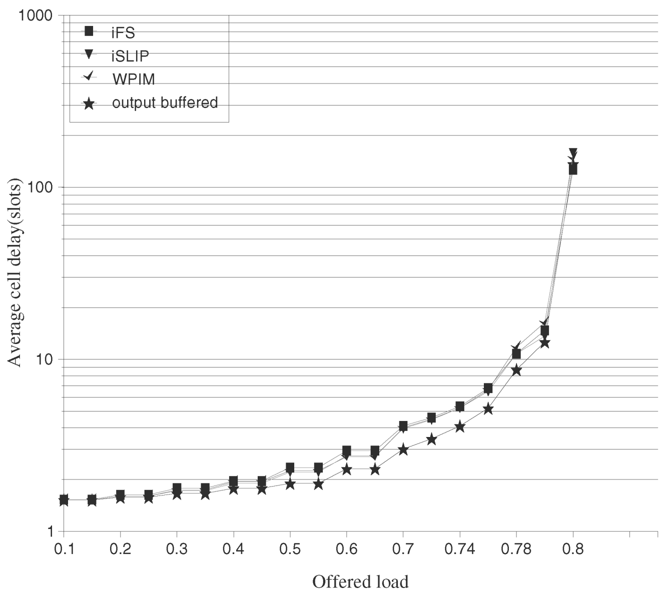


Fig. 5. Performance of *i*FS, *i*SLIP, and WPIM under nonuniform traffic.

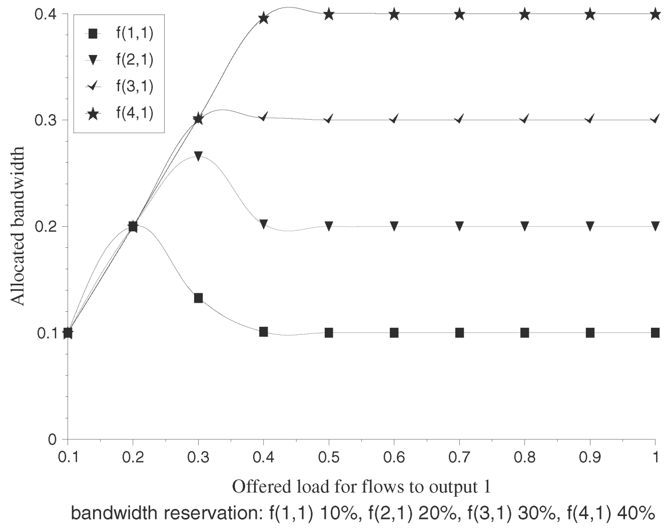


Fig. 7. Received bandwidth using *i*FS.

which states that the unused portion should be assigned equally to other active flows.

To quantify the fairness, we define *fairness index* in the following to measure the fairness of a switch for allocating bandwidth during time $(t_1, t_2]$. Suppose there are N flows sharing a link and let λ_i, x_i, and r_i denote the actual average arrival rate, received bandwidth, and reserved bandwidth, respectively, for flow i during time $(t_1, t_2]$. Without loss of generality, we assume that the first n_1 $(0 \leq n_1 < N)$ flows honor their reservation, that is, $\lambda_i \leq r_i$ for flows 0 through $n_1 - 1$. The rest of the $n_2 = N - n_1$ flows have arrival rate greater than their reservation. For each oversubscribing flow i $(n_1 \leq i < N)$, we denote

$$\beta_i = \frac{\Sigma_{j=0}^{n_1-1}(r_j - \lambda_j)r_i}{\Sigma_{k=n_1}^{N-1} r_k}. \qquad (1)$$

β_i is the bandwidth that can be spared to the oversubscribing flow i and the adjusted reservation for it is $r_i + \beta_i$. Ideally, the unused bandwidth is distributed proportionally to the backlogged flows and this is exactly what is conveyed in (1). Now, we define the fairness index (α) for a link l as

$$\alpha = \frac{\Sigma_{i=0}^{n_1-1}(|\frac{x_i-\lambda_i}{\lambda_i}|) + \Sigma_{i=n_1}^{N-1}(|\frac{x_i-min(\lambda_i,r_i+\beta)}{min(\lambda_i,r_i+\beta)}|)}{N}. \qquad (2)$$

α measures how close the received bandwidth is to the reservation for a link. The overall fairness index for a switch is calculated as the average α value of its output links:

$$\alpha_{switch} = \frac{\Sigma_{l=0}^{l=K-1}\alpha_l}{K}, \qquad (3)$$

where K is the number of outputs for the switch.

The smaller the α is, the better the fairness is. The fairness indices for the settings in Fig. 6 and Fig. 7 are 0.25 for *i*SLIP and 0.0185 for *i*FS at the input rate of 0.3. The α values are 0.573 and 0 for *i*SLIP and *i*FS, respectively, under the input rate of 0.4 or above.

The WPIM scheme also complies with the bandwidth requirement of each flow by restricting the number of transmitted cells during a frame within a limit determined by its reservation. Accordingly, the bandwidth requirement can be met and the well-behaved flows are protected. However, a careful study reveals the drawback of WPIM, whose mechanism rules that link bandwidth is evenly distributed among existing flows until some use up their quotas. Flows running out of their quotas are then excluded from accessing the link in the current frame and the bandwidth is again allocated equally among the remaining flows. Let us inspect the flow with the largest bandwidth reservation. Under WPIM, this flow shares the link bandwidth equally with all the other flows at the beginning of each frame. Its share increases gradually as the quotas for other flows are exhausted. As an example, consider the four flows going to output 1, but with bandwidth reservations of 40 percent, 20 percent, 20 percent, and 20 percent this time. Frame length is taken as 1,000 slots, as in [36]. We observe that every flow gets its fair share by the end of a frame under both WPIM and *i*FS. But, if we break a frame into four 250-slot subframes, we notice that the bandwidth

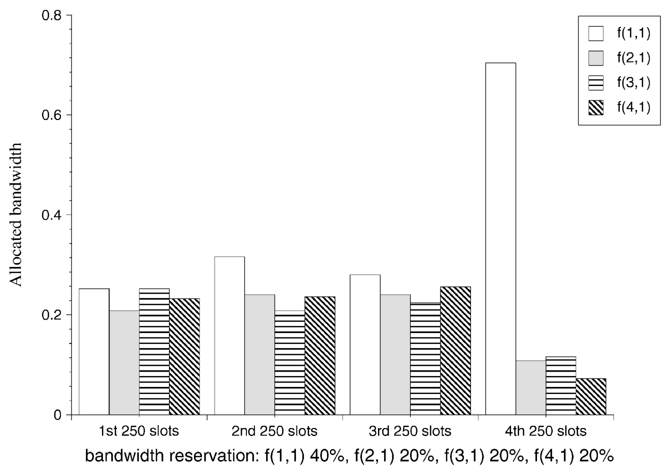


Fig. 8. WPIM: bandwidth distribution within a frame of 1,000 slots.

distribution for WPIM is not fair, as depicted in Fig. 8. During the first 750 slots, bandwidth is almost equally distributed (about 25 percent each), regardless of the reservation. Toward the end of the frame, flows with less reservation ($f(2,1)$, $f(3,1)$, and $f(4,1)$) have used up their quota. The flow with the highest reservation ($f(1,1)$) then consumes almost all the capacity in the last 250 slots. Therefore, WPIM provides bandwidth guarantee at coarse granularity of 1,000 slots. The *i*FS, on the other hand, can provide fair sharing both at the coarse and fine grain levels. The received bandwidth in *i*FS complies with the reservations even within subframes, as illustrated in Fig. 9. Their ability to support fair bandwidth is also compared by fairness indices in Table 1 for individual subframes. Some may argue that if the length of a frame in WPIM is small enough, it is able to support fair bandwidth sharing in finer granularity. But, we find that the WPIM scheme is "memoryless" regardless of the frame length since the scheduler does not have the information about the flow history except in the current frame. Imagine a scenario where the flow with the greatest reservation actually sends very little in the beginning; it cannot get "compensation" later when it raises the traffic. If it has nothing to send in prior frames, it loses its quota in those frames for good.

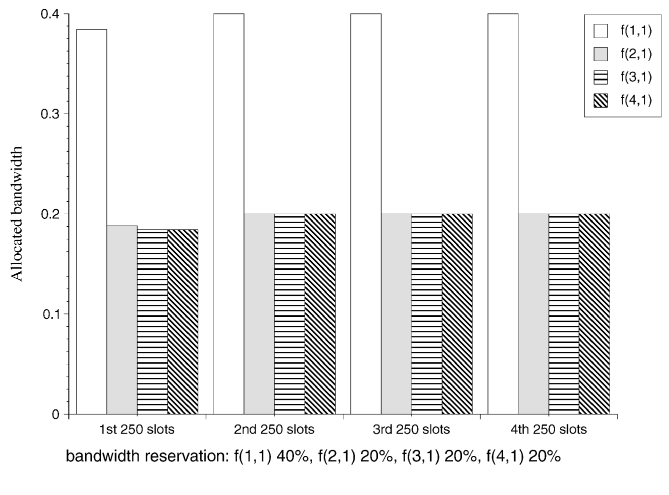


Fig. 9. *i*FS: bandwidth distribution within a frame of 1,000 slots.

TABLE 1
Fairness Indices for WPIM and *iFS* (SF for Subframe)

α	1st SF	2nd SF	3rd SF	4th SF
WPIM	0.21	0.158	0.225	0.568
iFS	0.015	0	0	0

While memoryless is necessary to enforce fairness, it is not desirable for WPIM with frames that are too short. Some fair queuing algorithms have been devised to compensate for such a flow to a certain extent [20].

As for all the iterative approaches, the number of iterations required for convergence is a constraining factor because all the iterations must be done within a single switch cycle. In the case of a 16×16 switch, 16 iterations are needed in the worst case. Fig. 10 shows the effect of the number of iterations on the average cell latency under uniform traffic. We can see that, with two iterations, a throughput of over 90 percent can be achieved and four iterations are adequate to obtain a throughput of nearly 100 percent. This is consistent with the findings in [1], [36], and [22] that $log(N)$ iterations are enough for the algorithm to converge.

In summary, the evaluation using uniform and nonuniform synthetic traffic models indicates that the *i*FS scheme is very promising in supporting fair bandwidth distribution, as well as in maintaining high overall throughput.

4.2 Measurement from Real Traffic

Studies using Internet traffic traces have been extensively reported in network research. Yet, to the best of our knowledge, ours is the first attempt to incorporate such traces into evaluating switch/router scheduling schemes. We hope this practice can shed some light on the effective use of real traces in validating the performance of different approaches.

Traffic arrivals in the Internet have been shown to be highly correlated (self-similar) [31], [7]. Simulations using Poisson or Bernoulli distribution offer good judgment on how a scheme works, but may significantly underestimate the burstiness of the real traffic pattern and give rise to unrealistic performance results. Geometrically distributed ON/OFF traffic is used in [22] and [13] to model the burstiness, but we will show later that the approach of taking real traces is better in characterizing the impact of traffic on switch design.

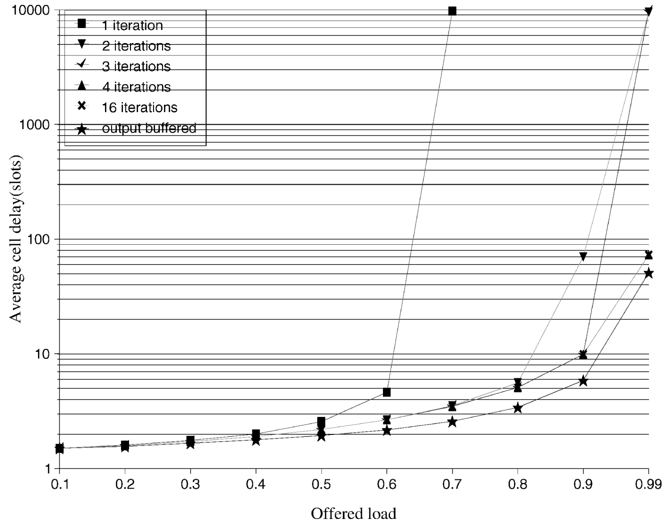


Fig. 10. Latency versus the number of iterations for *i*FS.

TABLE 2
Input Link Utilization for the Various Flows (NLANR)

utilization	link 1	link 2	link 3	link 4
overall	19.5%	20.5%	19.3%	18.6%
$f(i,1)$	7.6%	8.5%	18.9%	13.1%
$f(i,2)$	4.2%	4.5%	0.2%	3.5%
$f(i,3)$	4.7%	3.6%	0.09%	1.4%
$f(i,4)$	3.0%	3.8%	0.07%	0.6%

4.2.1 Measurement from NLANR Traces

The traces taken from the National Lab of Applied Network Research (NLANR [27]) are collected using OC3mon, a traffic monitor on an OC-3 link, at the ATM backbones on NSF vBNS. Traces from the same site were used in [16] for studying the packet forwarding method. The traces considered in this paper were collected in May 2000 from the facilities AIX, FRG, and MRT. Every line in the trace file includes the following information: timestamp when a TCP header arrives at the OC3mon, source IP/port, destination IP/port, and size of the packet. We use four traces as arriving traffic on four input links to our 4×4 switch. The overall average utilization for these input links is listed in the first row of Table 2. Based on the destination IP address, we classify traffic on each link into four flows, one for each output port (i.e., flow $f(i,j)$ from link i going to output j). The link utilization for the individual flows are also listed in Table 2. We derive each burst length from packet size as they are segmented into ATM cells. Each idle period is calculated as the interarrival time between two packet headers minus the burst length of the prior packet. Unfortunately, the traces carry no information about the bandwidth reservation so we impose our assumption of reserved bandwidth in the simulation.

To see how real traffic is different from geometrically distributed on/off traffic, we give an example by plotting cell arrival patterns for input link 1 and their corresponding geometric counterpart in Fig. 11 and Fig. 12, respectively. The geometrical on/off traffic is so generated that it has the same average burst length and idle period as those from the trace. We observe that the *on* time is much more clustered in the trace and the typical number of cells coming within 1,000 cycles ranges from 100 to 600. There are also nonnegligible times when not a single cell shows up during a 1,000-cycle period. For a geometrically distributed on/off arrival pattern, on the other hand, the interarrivals are more evenly spread out, with only 100 to 350 cells incoming within every 1,000 cycles. Studies from other links tell of similar differences between traces and synthetically generated traffic.

To examine the impact of the distinct traffic pattern, we feed the flows from the traces and the geometrically

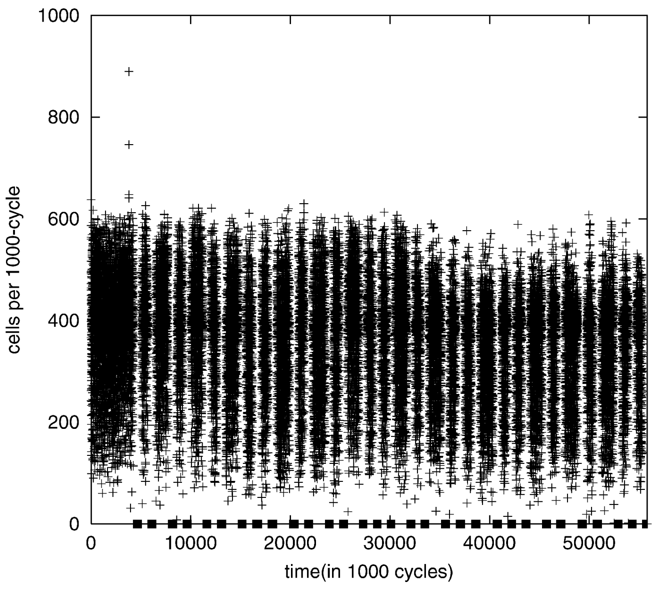


Fig. 11. The arrival pattern for traffic from link 1 (NLANR).

distributed on/off traffic to the simulator. Fig. 13 presents the delay for flows to output 1 under both cases. The average cell delay is substantially greater for the real traffic. Geometrically distributed on/off traffic underestimates the contention of the flows and therefore cannot be used very well as a representative model.

Now, we study effectiveness of the *i*FS scheme for maintaining fair bandwidth allocation for real traffic. On top of the flows presented in Table 2, we impose a bandwidth requirement arbitrarily. Let the bandwidth requirement for flows $f(1,1)$, $f(2,1)$, $f(3,1)$, and $f(4,1)$ be 10 percent, 20 percent, 30 percent, and 40 percent, respectively. As the sum of the workload is well below the capacity of output link 1, throughput for each flow is equal to the offered input rate. However, since some flows reserve a greater fraction of bandwidth than others, these flows should receive a larger bandwidth and perceive less delay. Fig. 14 shows the average cell delay for the flows to output 1 under *i*FS and *i*SLIP schemes. Note that *i*SLIP does not observe the bandwidth requirement. Flows $f(1,1)$ and $f(2,1)$ have lower delay because their input rate is only about 8 percent. Flow $f(3,1)$ experiences a much longer delay than $f(4,1)$ since it has a higher input rate than the latter (18.9 percent versus 13.1 percent). In contrast with the situation for *i*SLIP, the delay for the individual flows in *i*FS depends not only on the input rate, but also on the bandwidth requirement. It is shown in Fig. 14 that flow $f(1,1)$ has much greater average cell latency than that of $f(2,1)$, even though their input rates are close. This is because $f(1,1)$ reserves only 10 percent of the bandwidth, whereas 20 percent is set aside for $f(2,1)$. For the same reason, $f(3,1)$ suffers a larger delay than $f(4,1)$ since it has

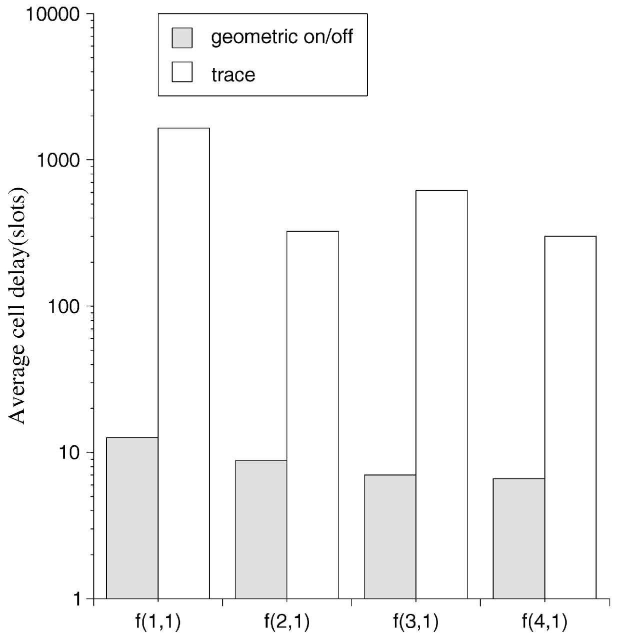


Fig. 13. Average cell delay for trace vs. geometrically on/off traffic.

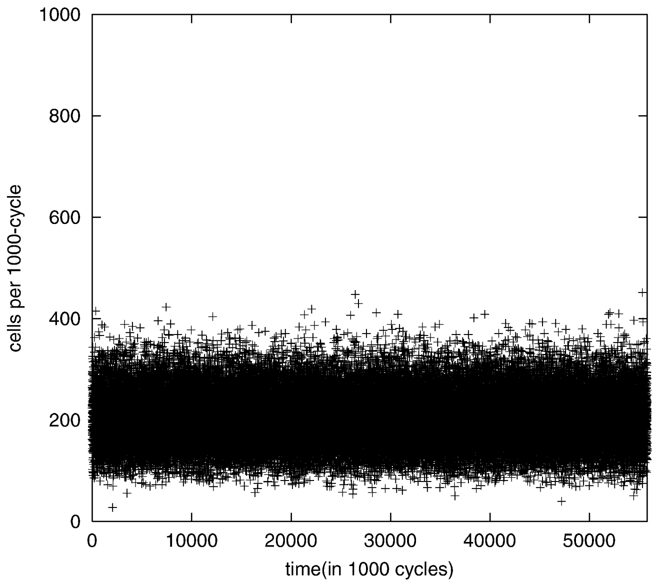


Fig. 12. The arrival pattern for geometric traffic from link 1.

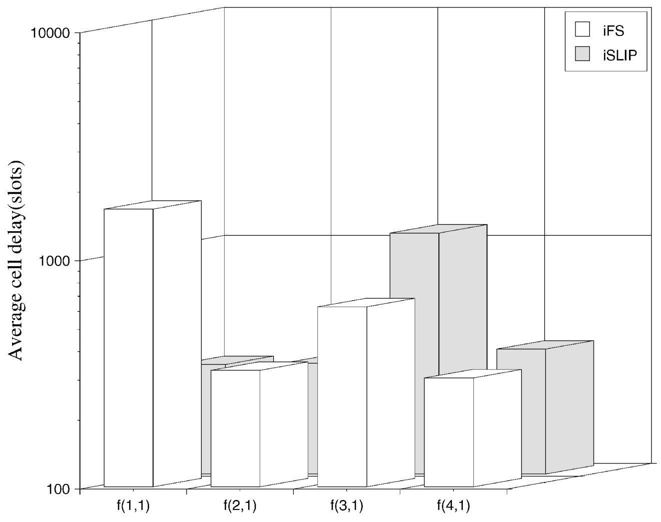


Fig. 14. Internet traffic: average cell delay for *i*FS and *i*SLIP.

TABLE 3
Link Utilization for the Flows after Cutting Down Idle Time (NLANR)

utilization	link 1	link 2	link 3	link 4
$f(i,1)$	24.7%	27.1%	48.3%	37.6%
$f(i,2)$	15.0%	15.8%	0.78%	12.8%
$f(i,3)$	16.6%	13.2%	0.35%	5.6%
$f(i,4)$	10.9%	13.5%	0.28%	2.2%

TABLE 4
Link Utilization for the Flows after Cutting Down Idle Time (UCB)

utilization	link 1	link 2	link 3	link 4
$f(i,1)$	23.3%	25.9%	36.1%	35.9%
$f(i,2)$	8.0%	22.1%	21.1%	14.0 %
$f(i,3)$	5.1%	19.9%	20.5%	15.8%
$f(i,4)$	28.5%	9.3%	20.5%	16.4%

a higher input rate (18.9 percent versus 13.1 percent), but reserves less bandwidth (30 percent versus 40 percent). However, a greater reservation itself cannot guarantee lower delay. Look at $f(2,1)$ and $f(3,1)$ for a case in point. Compared to $f(2,1)$ with a reservation of 20 percent, $f(3,1)$ has a larger average cell delay even if it reserves 30 percent of the bandwidth. The actual input workload for $f(3,1)$ is so much higher than $f(2,1)$ that its reservation is not great enough to bring in a lower delay.

In order to demonstrate how *i*FS enforces fair bandwidth assignment, we need flows to send packets at a higher rate than reservation. However, the highest link utilization from the traces available at NLANR is only around 20 percent and it becomes even lower after being split over four outputs. We get around this problem by cutting down each idle period to 25 percent of the original time in the simulation. The resulting average link utilization of various flows is listed in Table 3. We believe this manipulation on the traces will not compromise the quality of the real traffic. Assume the same bandwidth reservations of 10 percent, 20 percent, 30 percent, and 40 percent for $f(1,1)$, $f(2,1)$, $f(3,1)$, and $f(4,1)$, respectively. The achieved bandwidth of the flows is plotted in Fig. 15 for *i*FS and *i*SLIP schemes. It is observed that *i*FS is capable of supporting fair bandwidth allocation when the workload exceeds the capacity of the shared link and the received bandwidth for the individual flows is proportional to the reserved share. The *i*SLIP fails to do so and distributes the bandwidth evenly among the various flows due to the mechanism of rotating priority.

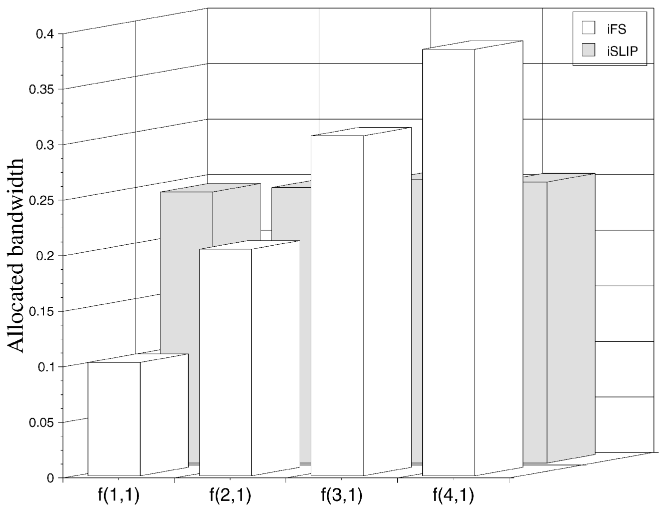


Fig. 15. Internet traffic: received bandwidth distribution.

4.2.2 Measurement from UCB Traces

Next, we consider Web proxy traffic from HTTP traces gathered by UC Berkeley in November 1996 from its Home IP service [37]. The community gains IP connection across about 600 modems (with speed from 2.4Kb/s to 28.8Kb/s) and all their traffic ends up going through a single 10Mb/s shared Ethernet segment on which a network monitoring computer is placed. Interesting fields from the traces include the time when the first byte of an HTTP response data file was seen, the time when the last byte of data was seen, anonymized client and server addresses, and the size of response data file. We ignore the effect of HTTP requests because their consumed bandwidth is negligible.

We do the following processing to the trace: According to the maximum transmission unit of the Ethernet, we segment each data file into a series of 1,500-byte packets. Since our simulator is cycle-based, each packet is further divided into cells, which are sent back-to-back as a burst. The idle time between two consecutive bursts is determined as if the packets belonging to a file are evenly spaced between the time of the first byte seen and the last byte seen. The idle period between the files is calculated directly from the trace. The interleaving of packet transmission from multiple files is also taken into account. Again, we assume a 4×4 switch and classify traffic depending on the IP address and then feed the traces into the four input links. Due to the same reason as for NLANR traces, the idle time is cut down to obtained higher link utilization, as shown in Table 4.

Assume the same bandwidth reservations of 10 percent, 20 percent, 30 percent, and 40 percent for $f(1,1)$, $f(2,1)$, $f(3,1)$, and $f(4,1)$, respectively. The achieved bandwidth of the flows from link 1 is plotted in Fig. 16 for *i*FS and *i*SLIP schemes. As expected, with *i*FS scheduling, the more a flows reserves, the more bandwidth it is entitled to. But, one thing brought to our attention is that, although $flow(1,1)$ has an average link utilization of 23.3 percent and has reserved 10 percent of the link bandwidth, its perceived bandwidth is only 6.8 percent. We examine the traffic pattern for this flow and plot it in Fig. 17. It is observed that, in approximately the first one third of the time, there are almost no packet arrivals from $flow(1,1)$. So, $f(1,1)$ has nothing to send during that period. After that, its traffic increases dramatically. However, due to the reservation of 10 percent only, $flow(1,1)$ is restricted by the *i*FS from transmitting too much because other completing flows are also over subscribed. Therefore, the resulting average received bandwidth over the entire monitored interval is only 6.8 percent. With *i*SLIP, $flow(1,1)$ grasps 25 percent of the bandwidth in its active period and attains 15.8 percent of the link bandwidth on average, well above the preserved

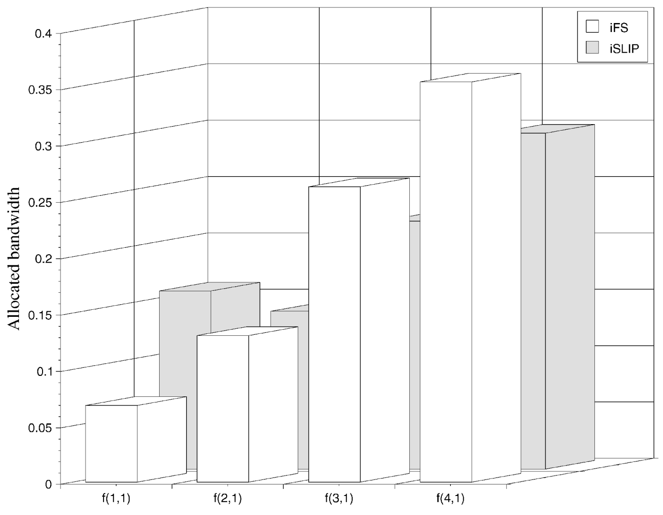


Fig. 16. Internet traffic: received bandwidth distribution (UCB).

10 percent. According to the trace collector, the uncharacteristically low activity in the traces corresponds to network outrages from Berkeley's ISP, rather than from trace failures [37]. For comparison, we plot an example of a normal flow arrival pattern in Fig. 18. We can see that, even under such a skewed scenario, *i*FS is able to distribute resources more fairly than *i*SLIP.

5 Buffer Management for Fair Scheduling

Section 3 focuses on the switch scheduling scheme itself, without considering the buffer size. Yet , in practice, the input buffer is finite. With rate-based flow control, which is the common choice for supporting bandwidth distribution, excessive packets are dropped when the buffer is full or congestion is anticipated. In the following, we study four selective packet discarding mechanisms and examine their impact on fair bandwidth allocation.

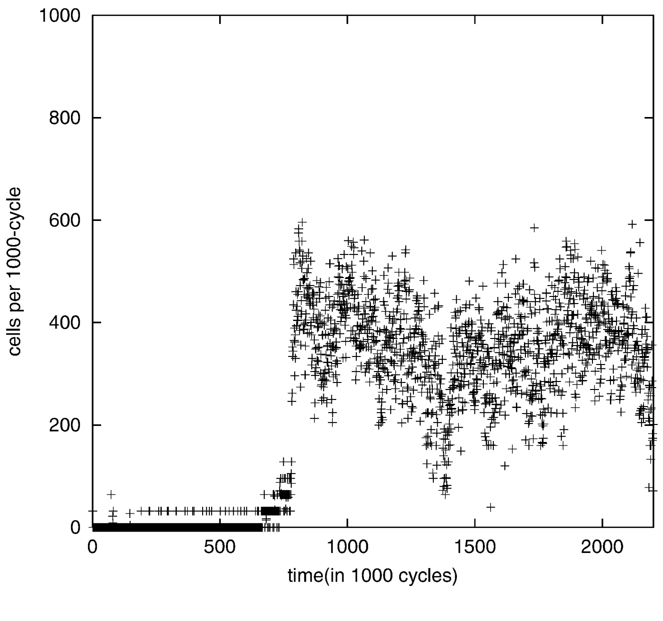


Fig. 17. The arrival pattern for traffic from flow (1, 1).

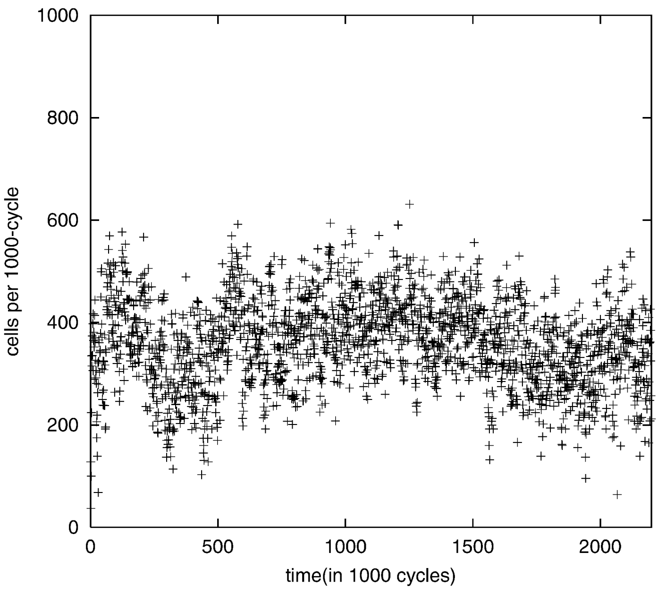


Fig. 18. The arrival pattern for traffic from flow (4, 1).

5.1 Decongestion Mechanisms

While messages from a source node are fragmented into fixed cells, which are transmitted individually across the network and reassembled at the destination, cell level retransmission is not supported. Thus, loss of a single cell in a packet forfeits the whole packet and it has to be retransmitted. Four decongestion mechanisms schemes are investigated: *drop tail* (DT), *early packet discard* (EPD), *equal size per flow* (ESPF), and *rate based size per flow* (RSPF). DT and EPD are stateless, whereas ESPF and RSPF are on a per-flow basis.

Drop tail: This is the basic dropping strategy: Cells are first-in-first-out and an incoming cell is dropped if it arrives to find the input buffer full. After a cell is shredded, the switch still makes an effort to transmit the remaining part of the packet, even if it turns out to be worthless at the destination and the entire packet has to be retransmitted. Hence, the DT scheme is poor in performance, despite its simplicity in implementation without keeping the status for each flow.

Early packet discard [11]: EPD overcomes the drawback of DT by dropping all the cells constituting a new packet when congestion is predicted. Upon receiving a header cell of a new packet, the switch checks to see whether the buffer occupancy exceeds a certain threshold. If so, the header cell is dropped and so are the upcoming cells from the same packet. Otherwise, the header is inserted into the buffer and subsequent cells are allowed into the buffer as long as it is not full upon their arrival. The justification behind this *early* discard is that, because the buffer is almost full and the congestion is likely to occur, the upcoming cells of the packet are most probably dropped. So, it is better to give up the packet that cannot be received successfully any way sooner than later. A good side effect is that, once the packet is dropped, it makes room in the buffer for other upcoming packets so that they are less likely to be discarded. Like the drop tail mechanism, EPD is also a stateless scheme. But, we need to watch for buffer occupancy constantly and keep track of the packets that have been selected for discard so that their upcoming cells are taken care of. Therefore, EPD is more intricate in implementation compared to DT. It will been seen later, with simulation, that the extra effort pays off.

Equal size per flow: Both drop tail and early packet discard are stateless and they are very much handicapped in the degree to which they achieve flow isolation, which is important in order to prevent ill-behaved flow from taking advantage of others. An EPD-based per flow queuing approach, called ESPF, secures an equal share of the buffer space after a certain threshold is reached. In ESPF, the share is calculated as the total buffer size divided by the number of concurrent active flows. Thus, this is a *dynamic* reservation policy in which the quota for each flow changes with the number of active flows. When buffer occupancy is below a certain threshold, all incoming cells are accepted. After that, a header cell is allowed into the buffer only if the flow's quota has not been used up. Otherwise, the entire packet is discarded. Once the header cell is admitted, subsequent cells can follow provided that the buffer is not full. One concern of ESPF in terms of implementation complexity is the necessity of computing the quota for the active flows on-the-fly because the flows are on and off.

Rate-proportional size per flow: ESPF attempts to assign an equal share of the buffer space to the active flows without taking their desired bandwidth into consideration. Intuitively, some flows may deserve a larger fraction of the buffer than others because they reserve a greater portion of the link bandwidth. In this sense, it is not necessarily fair to treat all the flows equally when allotting the buffer space. With the scheme of rate-proportional size per flow (RSPF), the quota is assigned in proportion to the fraction of the bandwidth reservation. As in ESPF, if buffer occupancy is low, all incoming cells are admitted. After the threshold is reached, a packet is allowed into the buffer only if its quota has not been exhausted. In contrast to ESPF, the quota for each flow is fixed by the time the reservation is made in RSPF. Therefore, it is less computationally intensive in implementation.

5.2 Performance Analysis

The decongestion mechanisms are evaluated based on the simulation with a 4×4 switch. From each of the four inputs, there are four flows going to different outputs, amounting to a total of 16 flows. We first examine the various schemes using geometric on/off traffic for average cell delay, packet loss ratio, and the ability to support fair bandwidth allocation. Then, we extend our study to employ real network traffic as input. The scheduling scheme used throughout this section is *i*FS and the threshold is chosen to be 80 percent of the buffer size.

Let us consider a scenario with a set of benign flows, where each of the 16 connections reserves an equal share of its intended link bandwidth and all the flows keep the same actual average traffic rate all the time. The workload is geometrically distributed on/off traffic and the average *on* period is 20 consecutive cells in a burst. The buffer size for each input block is set to 100 slots. We vary the offered input rate by changing the mean *off* time and examine the average cell delay and packet loss ratio, which is defined to

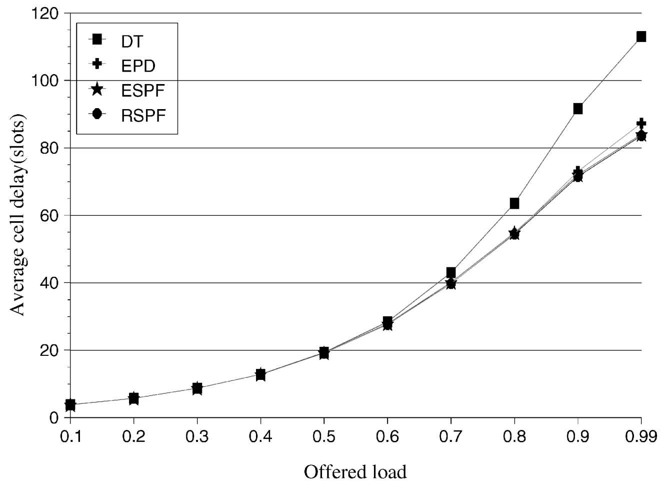


Fig. 19. Average cell delay with average burst length of 20.

be the ratio of the number of lost packets to total number of packets sent by the source. From the results in Fig. 19 and Fig. 20, it is observed that, under the input rate of 0.6, there is little difference among the four schemes. As the input rate further increases, the delay for DT is far worse than the others. In DT, incoming cells are dropped only when the buffer is full. Therefore, even if a header cell has already been discarded, constituent cells of the same packet may still clog in the buffer and compete to pass the crossbar. Such cells, eventually thrown away at the destination, worthlessly obstruct the way of useful cells from other packets and delay their transmission. The other three schemes all employ an early packet discard technique where useless cells are detected at an early stage and dropped so that delivery of other packets is expedited. Since all the flows reserve an equal share of bandwidth, RSPF becomes almost identical to ESPF under this circumstance, so it is not a surprise that ESPF and RSPF give nearly the same performance. However, careful study reveals that ESPF offers slightly lower packet loss ratio. We trace this to

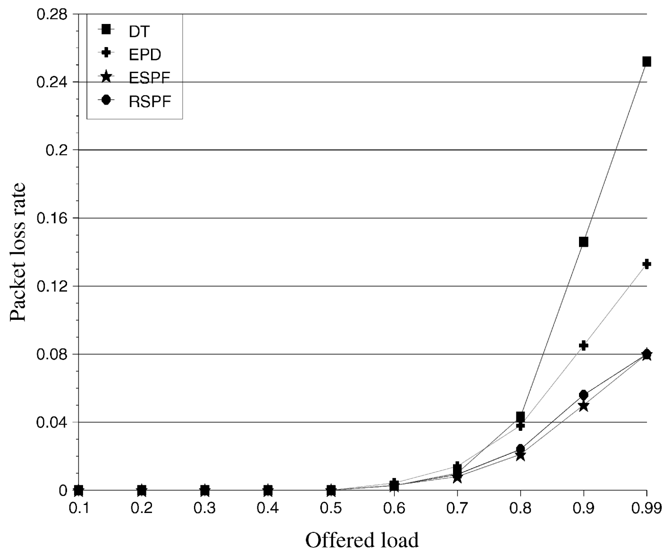


Fig. 20. Packet loss ratio with average burst length of 20.

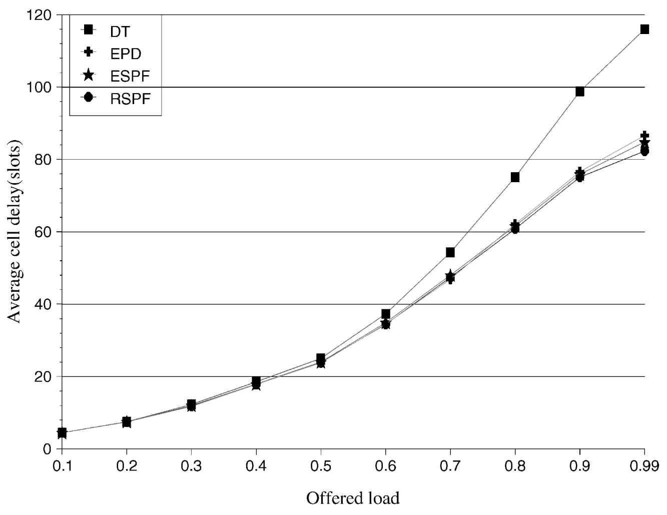


Fig. 21. Average cell delay with average burst length of 30.

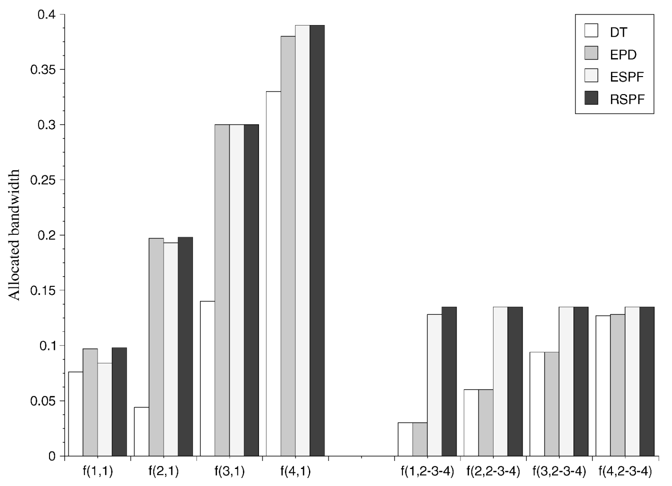


Fig. 23. The ability for fair bandwidth allocation under synthetic workload.

the fact that the buffer quota is dynamically assigned to the flows in ESPF. The active flows in ESPF can take advantage of the *off* flows and allow more packets in the buffer than in the case of RSPF.

Both ESPF and RSPF outperform EPD under heavy workload. With EPD, a new packet is discarded when the threshold is reached, whereas, in ESPF and RSPF, an incoming packet can still be admitted provided that the quota for its flow has not been exhausted. Therefore, with the same threshold, ESPF and RSPF allow more packets into the buffer. And, it explains why EPD has a much higher packet loss ratio than ESPF and RSPF in Fig. 20. It is possible to set a higher threshold in order for EPD to achieve better buffer utilization, but, again, the same problem with DT may occur.

In the next setting, we increase the average *on* period to 30 cells per burst and rerun the simulation. The resulting delay and packet loss rate are shown in Fig. 21 and Fig. 22. The trend is identical to the previous scenario, but, still, differences have been observed. For this setting, the delay under any given scheme is greater than that of the same scheme in the previous scenario when the average burst period is shorter. Comparing Fig. 19 and Fig. 21, the distinction is especially clear when the input rate is in the medium or upper higher range, within 0.4-0.8. In this range, incoming cells in the latter case find more cells accumulated in the buffer upon arrival, resulting in greater delay. With shorter bursts, on the other hand, the buffer is more likely be drained and, therefore, waiting time is less. Compared with Fig. 20, Fig. 22 shows a greater packet loss ratio for any given scheme. For a certain input rate, we find fewer number of packets in the long burst case. Therefore, a one-cell loss has greater impact on packet loss ratio in this case. Another observation is that the DT scheme is more sensitive to burstiness than others. At input rate 0.99, loss ratio increases from 0.25 to 0.36, while there is only a 0.03 increase for the rest of the schemes.

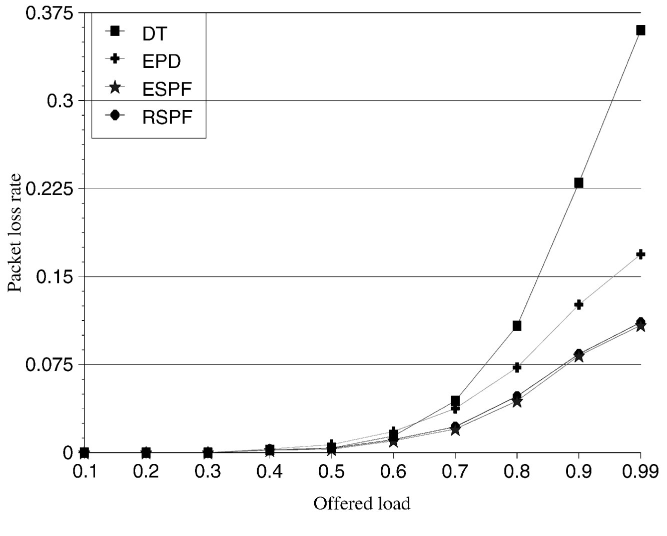


Fig. 22. Packet loss ratio with average burst length of 30.

The performances of the various decongestion mechanisms are inspected in supporting fair bandwidth distribution using geometric on/off workload. Following the notation of the tagged flows in Section 4, it is assumed $f(1,1)$, $f(2,1)$, $f(3,1)$, and $f(4,1)$ reserve 10 percent, 20 percent, 30 percent, and 40 percent of the output link 1 bandwidth, respectively. Each of the rest of the flows reserves a fraction of 15 percent of their intended link bandwidth. Suppose that the tagged flows fail to keep the contract and each sends at a rate of 0.4, but others abide their promises. The perceived bandwidth for the flows is presented in Fig. 23. Without flow isolation, DT performs the worst and cannot support bandwidth distribution according to the reservation. For example, compared to their respective reservation of 10 percent and 20 percent, $f(1,1)$ receives 7.6 percent and $f(2,1)$ gets 4.4 percent. Yet, the figure shows that the various flows do receive somewhat different bandwidth using DT. This is attributed to the fair scheduling scheme being used, which attempts to favor flows with greater reservation. EPD improves over DT by supporting fair bandwidth distribution among the tagged flows. However, if we look at untagged flows from input links 1 and 2, they receive bandwidth well below the reservation. The reason is that the actual sending rate from tagged flows of input 1 and 2 is so much higher than their

TABLE 5
Effect of Decongestion Mechanism on Fairness under Synthetic Workload

α	DT	EPD	ESPF	RSPF
link 1	0.420	0.024	0.055	0.014
link 2,3,4	0.480	0.480	0.110	0.100
switch overall	0.465	0.366	0.096	0.079

reservation that many of their packets are backlogged. Since EPD is not per-flow based, such packets clog the buffer and prevent cells from the untagged flows from getting in. Eventually, packets from the untagged flows are discarded, resulting in underutilization of their allocated bandwidth. With per-flow queuing, ESPF and RSPF sucessfully protect the benign flows in this scenario. The fairness indices for the four policies are given in Table 5. One may wonder why ESPF, without taking any reservations, works almost equally as well as RSPF. It is difficult to derive the distribution for queue length analytically since it is a $G/G/1$ queuing system from the point of view of each flow. But, intuitively, because fair scheduling is employed, flows reserving a greater fraction of bandwidth also receive a higher service rate and cells are drained more quickly. As a result, such flows do not necessarily require a buffer size in proportion to the arrival rate. For instance, in an $M/M/1$ system, the average queue length is $\frac{\lambda}{\mu-\lambda}$, where λ and μ are arrival and service rate, respectively.

Finally, the various decongestion mechanisms are compared using the Web traces, as in Section 4.2. For NLANR packet header traces, the link utilization can be found in Table 3. The bandwidth reservation imposed is assumed to be the same with the last setting, i.e., 10 percent, 20 percent, 30 percent, and 40 percent for the tagged flows and 15 percent otherwise. The perceived bandwidth for the individual flows under different schemes is plotted in Fig. 24. A significant difference between ESPF and RSPF is seen, especially for flow $f(3,1)$, whose traffic is the most intensive and clustered. Rate-based buffer space allocation

TABLE 6
Effect of Decongestion Mechanism on Fairness under Real Workload (NLANR)

α	DT	EPD	ESPF	RSPF
link 1	0.500	0.140	0.220	0.140
link 2	0.293	0.200	0.023	0.000
link 3	0.310	0.247	0.079	0.058
link 4	0.240	0.180	0.043	0.036
switch overall	0.336	0.192	0.086	0.059

gives better results under real traffic when the incoming packets are highly bursty. Although flows with greater bandwidth reservation drain their cells faster under fair bandwidth scheduling, it may not be fast enough to offset the fact that their cells arrive in a dramatically clustered fashion. The fact that EPD provides a more fair bandwidth allocation than ESPF for output link 1 (α value 0.14 versus 0.22 in the first row of Table 6) also proves this to be true since flows with a higher input rate are allowed to take more buffer space in EPD but not in ESPF. Therefore, an equal share of the buffer is not capable of maintaining fairness for such flows and it is beneficial to provide a larger buffer for those with a highly bursty arrival pattern.

We assume the same bandwidth requirement for the flows in the UCB Web proxy traces and rerun the simulation using various decongestion policies. The results are presented in Fig. 25 and Table 7. We again found that both RSPF and ESPF offer better performance than EPD. RSPF outperforms ESPF for some flows and is a little bit inferior for others. We attribute this to the traffic characteristics of the individual flows.

The above experiments indicate that the decongestion mechanism can significantly affect the switch performance and fair scheduling scheme alone cannot guarantee fairness. Early packet discard, which drops worthless packets at an earlier stage and relieves the network congestion, is a must. Flow isolation is important to support fair link bandwidth distribution. Under situations when the traffic is not highly bursty, equal sharing of buffer space is sufficient if the fair

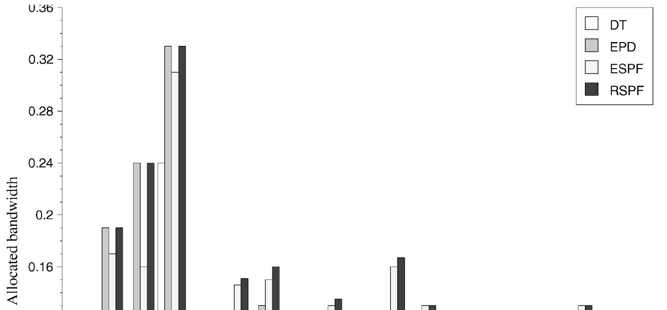


Fig. 24. The ability for fair bandwidth allocation for Internet traffic (NLANR).

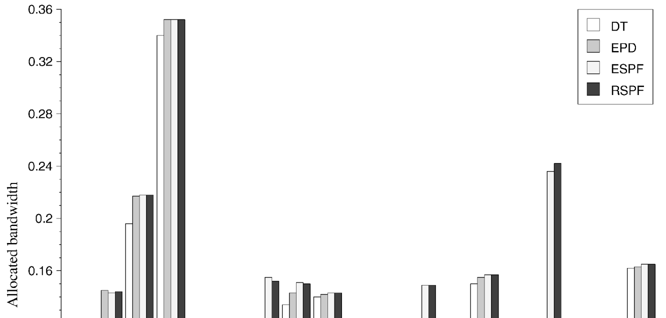


Fig. 25. The ability for fair bandwidth allocation for Internet traffic (UCB).

TABLE 7
Effect of Decongestion Mechanism on Fairness under Real Workload (UCB)

α	DT	EPD	ESPF	RSPF
link 1	0.400	0.270	0.240	0.250
link 2	0.440	0.330	0.170	0.174
link 3	0.460	0.320	0.160	0.160
link 4	0.508	0.240	0.260	0.230
switch overall	0.468	0.290	0.207	0.203

scheduling scheme is used. However, when cell arrival is highly clustered, equal buffer sharing will force the flows with high input rate to drop more packets. This can be avoided to a large extent if buffer allocation is in proportion to the bandwidth reservation.

6 Concluding Remarks

In this paper, we explored the scheduling schemes for input buffered switches to support fair bandwidth allocation. We first proposed an iterative fair scheduling (*i*FS) algorithm capable of scheduling cells so that each flow receives bandwidth proportional to its reservation under heavy traffic. We showed that fairness support does not compromise the average cell latency when compared with other iterative scheduling schemes. We examined four decongestion mechanisms and studied their impact on supporting fair bandwidth scheduling. With the extensive experimental results, we showed that early packet discard is necessary to relieve congestion. Our simulation also tells us that the fair scheduling scheme alone cannot ensure fair bandwidth allocation and per flow buffering is needed to protect well-behaved flows. Buffer allocation based on the bandwidth reservation offers better fairness, especially when the workload is highly clustered. It is worth mentioning that we explored the issues of switch design by incorporating the Web traffic traces in the study. Although it has been applied in other aspects of network research, it has not been used in the study of this area before. Results from a real trace workload provide further validation in addition to the commonly employed traffic model.

Acknowledgments

The authors would like to thank the referees for their helpful comments. This research was supported by US National Science Foundation Grant CCR 0105676. Another version of this paper was presented at IEEE INFOCOM 2002.

References

[1] T.E. Anderson, S.S. Owicki, J.B. Saxe, and C.P. Thacker, "High Speed Switch Scheduling for Local Area Networks," *ACM Trans. Computer Systerms,* vol. 11, no. 4, pp. 319-352, Nov. 1993.

[2] M. . Arlitt and L. Williamson Carey, "Internet Web Servers: Workload Characterization and Performance Implications," *IEEE/ACM Trans. Networking,* vol. 5, no. 5, pp. 631-645, Oct. 1997.

[3] G. Armitage and K. Adams, "Packet Reassembly during Cell Loss," *IEEE Network Magazine,* vol. 7, no. 5, pp. 26-34, Sept. 1993.

[4] P. Barford and M. Crovella, "Generating Representative Web Workloads," *Proc. 1998 ACM SIGMETRICS Int'l Conf. Measurement and Modeling of Computer Systems,* pp. 151-160, July 1998.

[5] J. Bennett and H. Zhang, "WF^2Q: Worst-Case Fair Weighted Fair Queueing," *Proc. IEEE INFOCOM 96,* pp. 120-128, Mar. 1996.

[6] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-Like Distributions: Evidence and Implications," *Proc. IEEE INFOCOM '99,* pp. 126-134, Mar. 1999.

[7] M.E. Crovella and A. Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes," *IEEE/ACM Trans. Networking,* vol. 5, no. 6, pp. 835-846, Dec. 1997.

[8] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *J. Internetworking Research and Experience,* vol. 1, no. 1, pp. 3-26, Sept. 1990.

[9] J. Ding and L.N. Bhuyan, "Evaluation of Multi-Queue Buffered Multistage Interconnection Networks under Uniform and Non-Uniform Traffic Patterns," *Int'l J. Systems Science,* vol. 28, no. 11, pp. 1115-1128, 1997.

[10] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Trans. Networking,* vol. 1, pp. 397-413, Aug. 1993.

[11] S. Floyd and A. Romanov, "Dynamics of TCP Traffic over ATM Networks," *IEEE J. Selected Areas in Comm.,* vol. 13, no. 4, pp. 633-641, 1995.

[12] L. Georgiadis, I. Cidon, R. Guerin, and A. Khamisy, "Optimal Buffer Sharing," *IEEE J. Selected Areas in Comm.,* vol. 13, pp. 1229-1240, Sept. 1995.

[13] M.W. Goudreau, S.G. Kolliopoulos, and S.B. Rao, "Scheduling Algorithms for Input-Queued Switches: Randomized Techniques and Experimental Evaluation," *Proc. IEEE INFOCOM '00,* vol. 3, pp. 1634-1643, Mar. 2000.

[14] P. Goyan, H. Vin, and H. Chen, "Start-Time Fair Queuing: A Scheduling Algorithm for Integrated Services," *Proc. ACM SIGCOMM,* pp. 157-168, Aug. 1996.

[15] P. Gupta and N. McKeown, "Design and Implementation of a Fast Crossbar Schedule," *IEEE Micro,* pp. 20-28, Jan./Feb. 1999.

[16] F. Hoymany and D. Mosse, "A Simulation Study of Packet Forwarding Method over ATM," *Proc. IEEE INFOCOM '98,* pp. 401-408, Mar. 1998.

[17] M.J. Karol, M.G. Hluchyj, and S.P. Morgan, "Input versus Output Queueing on a Space-Division Packet Switch," *IEEE Trans. Comm.,* vol. 35, no. 12, pp. 1347-1356, Dec. 1987.

[18] S. Keshav and R. Sharma, "Issues and Trends in Router Design," *IEEE Comm. Magazine,* pp. 144-151, May 1998.

[19] S. Li and N. Ansari, "Input-Queued Switching with QoS Guarantees," *Proc. IEEE INFOCOM '99,* pp. 1152-1159, Mar. 1999.

[20] S. Lu, V. Bharghavan, and R. Srikant, "Fair Scheduling in Wireless Packet Networks," *IEEE/ACM Trans. Networking,* vol. 7, no. 4, pp. 473-489, Aug. 1999.

[21] A. Mahanti, C. Williamson, and D. Eager, "Traffic Analysis of a Web Proxy Caching Hierarchy," *IEEE Network Magazine,* pp. 16-23, May/June 2000.

[22] N. McKeown, "The iSLIP Scheduling Algorithm for Input-Queued Switches," *IEEE/ACM Trans. Networking,* vol. 7, no. 2, pp. 188-201, Apr. 1999.

[23] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% Throughput in an Input-Queued Switch," *Proc. IEEE INFOCOM '96,* pp. 296-302, Mar. 1996.

[24] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz, "Tiny Tera: A Packet Switch Core," *IEEE Micro,* pp. 26-33, Jan./Feb. 1997.

[25] C. Metz, "IP Routers: New Tool for Gigabit Networking," *IEEE Internet Computing,* pp. 14-18, Nov./Dec. 1998.

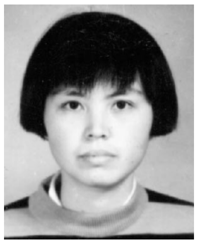
[26] P. Misha and M. Saksena, "Designing Buffer Management Policies at an IP/ATM Gateway," *Proc. IEEE ATM Workshop,* pp. 144-153, 1998.

[27] Nat'l Lab for Applied Network Research (NLANR), "Packet Header Traces," http://moat.nlanr.net/traces/, 2000.

[28] G. Nong and M. Hamdi, "On the Provision of Quality-of-Service Guarantees for Input Queued Switches," *IEEE Comm. Magazine,* pp. 62-69, Dec. 2000.

[29] 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.

[30] C. Partridge, P.P. Carvey, E. Burgess, I. Castineyra, T. Clarke, L. Graham, M. Hathaway, P. Herman, A. King, S. Kohalmi, T. Ma, and J. Mcallen, "A 50-Gb/s IP Router," *IEEE/ACM Trans. Networking,* vol. 6, no. 3, pp. 237-248, June 1998.
[31] V. Paxson and S. Floyd, "Wide Area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Trans. Networking,* vol. 3, no. 3, pp. 226-244, June 1995.
[32] L.L. Peterson and B.S. Davie, *Computer Networks—A System Approach.* San Francisco: Morgan Kaufmann, 2000.
[33] M. Pirvu, N. Ni, and L.N. Bhuyan, "Exploring the Switch Design Space in a CC-NUMA Multiprocessor Environment," *Proc. Int'l Parallel and Distributed Processing Symp.,* pp. 703-710, May 2000.
[34] D. Serpanos and P. Antoniadis, "FIRM: A Class of Distributed Scheduling Algorithm for High-Speed ATM Switches with Multiple Input Queues," *Proc. IEEE INFOCOM '00,* vol. 2, pp. 548-555, Mar. 2000.
[35] D.C. Stephens and H. Zhang, "Implementing Distributed Packet Fair Queueing in Scalable Switch Architecture," *Proc. IEEE INFOCOM '98,* pp. 282-290, Mar. 1998.
[36] D. Stiliadis and A. Varma, "Providing Bandwidth Guarantees in an Input-Buffered Crossbar Switch," *Proc. IEEE INFOCOM '95,* pp. 960-968, Apr. 1995.
[37] UC Berkeley Home IP Web Traces, http://ita.ee.lbl.gov/html/contrib/UCB.home-IP-HTTP.html, 2000.
[38] H. Zhang, "Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks," *Proc. IEEE,* vol. 83, pp. 1374-1396, Oct. 1995.



Nan Ni received the BS degree in computer science from Fudan University, the MS degree in computer engineering from Nanjing University, and the PhD degree in computer science from Texas A&M University. She is currently working in the Server Group at IBM Corporation. Her research interests include computer architecture, computer networks, I/O subsystems, and performance evaluation. She is a member of the IEEE.

Laxmi Narayan Bhuyan received the PhD degree in computer engineering from Wayne State University in 1982. He has been a professor of computer science and engineering at the University of California, Riverside since January 2001. Prior to that he was a professor of computer science at Texas A&M University (1989-2000) and program director of the Computer System Architecture Program at the US National Science Foundation (1998-2000). He has also worked as a consultant to Intel and HP Labs. His research interests are in the areas of computer architecture, parallel and distributed processing, interconnection networks, and performance evaluation. He has published more than 100 papers in these areas in the *IEEE Transactions on Computers*, *IEEE Transactions on Parallel and Distributed Systems*, *Journal of Parallel and Distributed Computing*, and many refereed conference proceedings. Dr. Bhuyan is a fellow of the IEEE and a fellow of the ACM.