

Max-Min Fair Input-Queued Switching

Madhusudan Hosaagrahara
and
Harish Sethu

Technical Report DU-CS-05-11
Department of Computer Science
Drexel University
Philadelphia, PA 19104
August 2005

Max-Min Fair Input-Queued Switching

Madhusudan Hosaagrahara and Harish Sethu

Abstract

Fairness in traffic management can improve the isolation between traffic streams, offer a more predictable performance, eliminate transient bottlenecks, mitigate the effect of certain kinds of denial-of-service attacks, and serve as a critical component of a quality-of-service strategy to achieve certain guaranteed services such as delay bounds and minimum bandwidths. While fairness in bandwidth allocation over a shared output link has been studied extensively, the desired eventual goal is overall fairness in the use of all the resources in a switch or a router. Several research efforts have attempted to achieve fairness in input-queued switches by one measure or another. However, there has been no uniform framework for evaluating these different strategies. In this report, we formalize the notion of fairness in input-queued switches through a rigorous framework that permits a quantitative evaluation of various strategies for fairness. Based on an extension of the max-min notion of fairness, our framework permits the algorithmic determination of the fair rates to be allocated to the flows through the switch. In addition to proving the correctness of the algorithm, we show that it can be implemented in a distributed fashion to dynamically determine flow rates. We further propose a practical scheduling strategy based on our algorithm. We present simulation results, using both real traffic traces as well synthetic traffic, to evaluate the fairness of a variety of popular scheduling algorithms for input-queued switches. The results show that our scheduling strategy achieves better fairness than other known algorithms for input-queued switches.

1 Introduction

1.1 Motivation

Flows of traffic through a network share a variety of resources in their path. Fairness in the allocation of resources in a network shared amongst multiple users is not only an intuitively desirable goal but also one with many practical benefits. Fairness in traffic management has traditionally been advocated to isolate and protect the performance of a flow from other flows that misbehave either due to software errors or deliberate misuse. In addition, strategies and algorithms for fair management of network traffic can serve as a critical component of Quality-of-Service (QoS) mechanisms to achieve certain guaranteed services such as delay bounds and minimum bandwidths [1]. Fairness in traffic management reduces the burstiness of flows, eliminates certain kinds of bottlenecks, and offers a more predictable performance [2]. More recently, fair resource allocation strategies have also been proposed to help reduce the impact of certain kinds of denial-of-service (DoS) attacks [3–5].

Many formal notions of fairness, such as Proportional Fairness, Utility Fairness, General Weighted Fairness and Max-Min Fairness, have been proposed in the research literature for the allocation of a single shared resource among competing entities [6–10]. The *max-min* notion of fairness is among the more popular notions and is based on the simple idea that increasing the allocation of any entity should not result in a decrease in the allocation of another entity that received an equal or smaller allocation of the resource¹ [8].

The reference model for the max-min notion of fairness is the ideal but unimplementable Generalized Processor Sharing (GPS) discipline [11]. Many practical fair scheduling algorithms that seek to achieve an approximation of GPS have been proposed and successfully deployed in Internet routers and operating systems [1,12,13]. The notion of max-min fairness and the scheduling algorithms that try to achieve it have also been successfully applied to various other facets of networking including routing, load balancing, wireless and ad-hoc networks [14–16].

Fair scheduling algorithms developed for bandwidth allocation have typically assumed an output-queued switch with each set of competing flows sharing a single resource (the link bandwidth). In such a switch, packets arriving at the input ports are immediately transferred to their destination

¹A more rigorous definition of max-min fairness for a single resource is provided in Section 2.1 and for input-queued switches in Section 4.2.

output port and buffered if necessary. This requires that packet buffers at the outputs operate faster than the line rate since they must, in the worst case, buffer multiple packets arriving simultaneously from all the input ports. Flows share only their destination output port and there is no interference between flows destined to different output ports. Overall fairness in an output-queued switch can therefore be simply achieved by being fair at each output port. This simplicity, however, has become increasingly difficult to achieve with the advent of high-speed optical links which carry data at rates that are orders of magnitude higher than memory speeds. As a result, a significant trend has emerged toward the design and use of input-queued switches based on the virtual output queueing (VOQ) architecture. Packets arriving at the input ports of a VOQ switch are buffered at the input ports in different queues based on their destination output port, thus requiring the packet buffers to operate only at the line rate [17–19]. Unfortunately, it is non-trivial to extend the fair scheduling strategies developed for output-queued switches to VOQ switches. This report is concerned with extending and applying the notion of max-min fair allocation in the context of a VOQ switch.

Fair scheduling in input-queued switches is a multiple resource allocation problem with multiple flows contending for different subsets of a shared set of resources. We identify a flow in an input-queued switch by an input-output port pair and define it as comprising the traffic between these two ports. Some sets of flows share an input port and some other sets of flows share an output port. All flows share the crossbar between the input and the output ports. The crossbar is also a resource that flows compete for, since it typically limits the total data transfer rate out of any given input port or into any given output port.

Our motivation for this report derives from the fact that simply being fair at each input port or at each output port or at both input and output ports does not result in an overall fair allocation. We illustrate this using the traffic pattern depicted in Fig. 1(a) in a 4-input 4-output input-queued switch, where $\{A, B, C, D\}$ is the set of input ports and $\{W, X, Y, Z\}$ is the set of output ports. Assume that the flows are all best-effort and therefore have demands equal to the line rate assumed at 1 unit of bandwidth. For the sake of this example, we assume that each input port always has packets available to transmit along the traffic pattern shown in Fig. 1(a). We further assume that the crossbar can transfer data out of any given input port at no more than 1 unit of bandwidth, and into any given output port at no more than 1 unit of bandwidth, leading to the demand matrix shown in Fig 1(b).

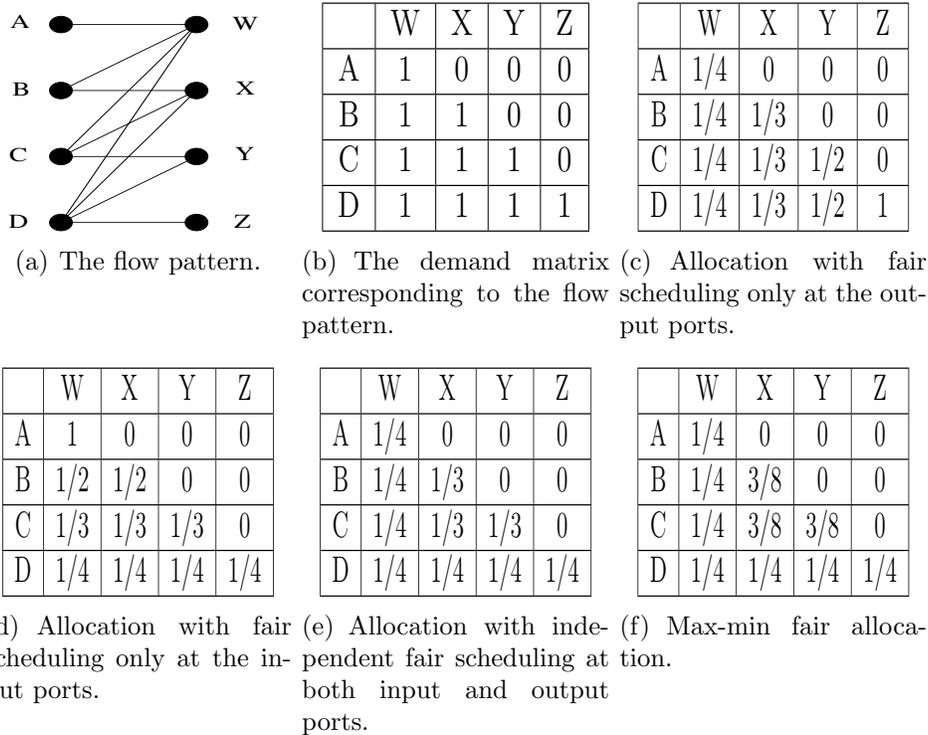


Figure 1: An illustration of why it is non-trivial to achieve fairness in input-queued switches.

Fig. 1(c) describes the allocations received by the flows when a max-min fair scheduling algorithm is used only at the output ports. In this case, input ports C and D are over-subscribed since the crossbar can transmit no more than 1 unit of bandwidth out of any given input port. This makes the crossbar a bottleneck resource. In the absence of a fair scheduling mechanism for bandwidth handled by the crossbar, there is no guarantee of overall fairness.

Fig. 1(d) describes the allocations received when a max-min fair scheduling algorithm is used only at the input ports. In this case, output ports W and X are over-subscribed. In the absence of any fair scheduling at the output ports or at the crossbar, there is no guarantee that flows heading for output ports W and X will receive their fair share of the bandwidth.

Applying fair scheduling at both input and output ports and without any communication between the schedulers results in an allocation illustrated

in Fig. 1(e). Here, a flow receives the lesser of its allocation at its input and output ports. This allocation does not cause a bottleneck at any input port, any output port, or at the crossbar. However, the allocations are not max-min fair, since the flows BX, CX and CY may increase their allocations without reducing the allocation of another flow with a smaller allocation. Such a strategy also results in under-utilization of available bandwidth since ports B and Y are not completely allocated. While we shall prove it later in section 4.3, Fig. 1(f) depicts the ideal max-min fair allocation since it is impossible to increase the allocation of any flow without decreasing the allocation of some other flow with a smaller allocation, while also satisfying the constraints of the crossbar.

The above begs the question of how one may compute the rates for max-min allocation in an input-queued switch and how one may achieve such an allocation through a practical strategy. Related work on achieving fairness in input-queued switches has not considered the problem within a rigorous framework founded on a formal notion of max-min fairness defined for input-queued switches.

1.2 Contributions

In this report, we extend the notion of max-min fairness to the context of input-queued switches and provide a rigorous definition of fairness in the allocation of multiple resources such as in input-queued switches. This definition serves as a framework to evaluate and compare the fairness of different scheduling strategies. We use this definition to introduce a new algorithm, called the *Fair Resource Allocation (FRA)* algorithm, that computes fair rates of allocation for each flow through the switch. FRA is an iterative algorithm which utilizes the concept of bottleneck links. We analytically prove that the FRA algorithm yields rates that are indeed max-min fair.

In addition, as opposed to previously known algorithms for computing the max-min rates [8], our algorithm lends itself very easily to a distributed implementation in an input-queued switch. We describe the distributed implementation of the FRA algorithm, called *Distributed FRA (DFRA)*, that can be combined with a variety of iterative scheduling algorithms used in input-queued switches. We describe a practical credit-based algorithm based on DFRA, which we call the *Largest Credit First (LCF)* algorithm, that achieves a very good approximation of the ideally fair rates.

We then present simulation results, based on both *real* and synthetic

traffic, comparing the fairness and performance of LCF with other popular scheduling strategies for input-queued switches. Our results indicate that LCF offers much better fairness than all the other schedulers, while performing only negligibly worse than the best scheduler.

1.3 Organization

The rest of this report is organized as follows. Section 2 discusses the background and provides a definition of the max-min notion of fairness and a brief overview of scheduling algorithms for input-queued switches. Section 3 describes related work on achieving fairness in input queued switches. Section 4 introduces the FRA algorithm which computes the ideally fair rates of flows and provides a proof of its correctness. Section 5 describes the distributed implementation of the FRA and presents strategies for practical algorithms based on DFRA. Section 6 presents simulation results that demonstrate the improved fairness achieved by our algorithm. Section 7 concludes the report.

2 Background

2.1 Max-Min Fairness

Max-Min is a popular notion of fairness in the allocation of a shared resource amongst competing entities with different demands [1, 8]. Consider a shared resource with the total amount of available resource denoted by R . A set of n entities contending for this resource have demands $\mathcal{D} = [d_1, d_2, \dots, d_n]$. Assume that all entities have equal rights to the shared resource and that an allocation policy \mathcal{P} is used to yield an allocation vector $\mathcal{A} = [a_1, a_2, \dots, a_n]$.

Definition 1 The allocation vector \mathcal{A} is said to be *feasible* if and only if [8]:

1. Each entity receives an allocation greater than or equal to zero; i.e., for all i , $a_i \geq 0$.
2. The total allocated resource is less than the available resource; i.e., $\sum_{\forall i} a_i \leq R$,

Definition 2 The allocation vector \mathcal{A} is said to be *max-min fair* if:

1. It is feasible.

2. No entity receives an allocation greater than its demand; i.e., for all i , $a_i \leq d_i$.
3. For all i , the allocation of entity i cannot be increased while satisfying the above two conditions and without reducing the allocation of some other entity j for which $a_j \leq a_i$.

The allocation policy P is said to be max-min fair if for any arbitrary number of users with arbitrary demands, it always yields a max-min fair allocation. We denote the operation of such a policy P as the MMF operator defined as follows:

Definition 3 Define the Max-Min Fair (MMF) operator as one that given a scalar constraint R and a vector of demands $\mathcal{D} = [d_1, d_2, \dots, d_n]$ yields an allocation vector that is max-min fair. In other words, if

$$\mathcal{A} = \text{MMF}(R, \mathcal{D}),$$

then, the allocation vector $\mathcal{A} = [a_1, a_2, \dots, a_n]$ is max-min fair.

Definition 4 An entity is said to be *satisfied* if the allocation it receives is equal to its demand (i.e., entity i is satisfied if $a_i = d_i$). Otherwise, the entity is said to be *unsatisfied*.

Note that, in the case of a single shared resource, the following are some of the properties of any max-min fair allocation policy, and consequently, of the MMF operator.

Property 1 No entity is allocated a share of the resource larger than that of another entity with the same or larger demand.

Property 2 Two entities that are unsatisfied receive equal shares of the resource. In other words, if $a_k < d_k$ and $a_l < d_l$, then $a_k = a_l$.

Property 3 An unsatisfied entity cannot increase its allocation by simply increasing its demand.

The above notion of max-min fairness has been most commonly applied in algorithms for scheduling packets for transmission over a shared output link [1].

2.2 Input-Queued Switches

In an input-queued switch based on virtual output queueing, packets arriving at the input ports are buffered based on their destination output port at the input port itself, thus requiring the packet buffers to operate only at the line rate. Recent research has shown that it is possible to achieve 100% throughput as well as support Quality-of-Service (QoS) guarantees using input-queued switches [20]. Central to this success are the scheduling algorithms that control access to the switch fabric and match input ports to output ports.

Scheduling in an input-queued switch for best-effort traffic is a bipartite graph matching problem. The input ports and the output ports form the two disjoint vertex sets. The edge set of the graph is formed by adding an edge between an input port and an output port if the input port has a packet queued for transmission through the output port. The scheduling problem is to generate a subset of edges such that each vertex has at most one edge incident on it. This subset of edges describes the set of packets that should be transferred from the input ports to the output ports.

It is desirable that the largest subset of edges be selected to maximize the overall throughput of the switch. The fastest known algorithm for Maximum Size Matching (MSM) in bipartite graphs was proposed by Hopcroft and Karp [21]. Although the algorithm guarantees the largest matching possible and therefore maximizes the overall throughput, it is unfair since it may lead to starvation of flows (i.e., some input and output ports are never matched). In addition, it has been shown to be unstable even for admissible traffic patterns; i.e., even if none of the input and output ports are over-subscribed, a queue may not become empty with probability one after some finite time [22]. On the other hand, it has been proved that a maximum weight matching, with the weight function based either on the queue length (called the Longest Queue First or LQF algorithm) or on the queueing delay of the head-of-line packet (called the Oldest Cell First or OCF algorithm), is stable for all admissible traffic patterns [20]. It has also been shown that these algorithms can provide bandwidth and delay guarantees [17, 23, 24]. While these algorithms are stable and guarantee high-throughput, their implementation and time complexity make them unsuitable for implementation in high-speed switches.

The need for simple, efficient and fast schedulers for high-speed switches was first addressed by Anderson *et al.* The Parallel Iterative Matching (PIM)

algorithm is a distributed maximal matching algorithm based on iterative negotiations between input and output ports [25]. The PIM scheduling algorithm is composed of multiple iterations, with each iteration having three distinct phases.

- The *Request* Phase: In this phase, the input ports identify the output ports to which they have at least one packet queued. They then proceed to send a *Request* signal to these output ports, asking for permission to transmit.
- The *Grant* Phase: The output ports receive *Requests* from the input ports. Each output port randomly selects one of the requesting input ports and sends a *Grant* signal to the selected input port indicating a permission to transmit.
- The *Accept* Phase: The input ports receive the grants (an input port may receive more than one grant). Each input port randomly selects one of the grants and sends an *Accept* signal to the corresponding output port indicating acceptance of the grant.

The above three phases complete a single iteration. Ports between which an *Accept* signal was exchanged during an iteration are considered matched and they do not take part in subsequent iterations. Since PIM is a randomized algorithm, it cannot place a finite upper bound on the length of time a flow might be starved, and hence cannot provide fairness guarantees.

The *iSLIP* algorithm eliminates the use of randomness by issuing *Grants* and *Accepts* in a round robin fashion by the output and input ports, respectively [26]. A counter maintained by each port keeps track of the next port to which a grant or an accept is to be issued. Input ports increment the counter to one past the last accepted output port, while output ports increment their counter by one past the input port it granted to, if and only if the grant is accepted by the input port in the *Accept* phase. It has been shown that *iSLIP* can provide 100% throughput for uniform traffic [26].

Since the publication of PIM and *iSLIP*, numerous iterative schedulers for input-queued switches have been proposed. Most of these schedulers are randomized algorithms concentrating on achieving 100% throughput and simplicity of implementation, while providing only weak QoS guarantees such as a high probability of eliminating starvation.

3 Related Work

There have been a few attempts to achieve max-min fairness in input queued switches. These approaches have taken one of the following two approaches: emulating a GPS system and throttling. Schedulers that adopt the former approach are usually based on the Distributed Packet Fair Queueing architecture, which proposes implementing a fair scheduler at each of the input and output ports, along with a scheme to synchronize the virtual times of these schedulers [27, 28]. One practical implementation of this approach is the Iterative Port-based Deficit Round Robin (IPDRR) scheduler which is an adaptation of the Deficit Round Robin (DRR) scheduler to input-queued switches [18, 29]. Each port uses the DRR scheduler to achieve fair and guaranteed bandwidth allocation, while an iterative approach with *Grants* and *Accepts*, similar to PIM and *iSLIP*, is used to schedule the packets. The IPDRR scheduler supports guaranteed rates without requiring speedup and has a time-complexity of $O(1)$. Another proposed scheduler adopting this approach is the min-max fair queueing which maintains two systems: a virtual system which exactly emulates a virtual GPS server and a real system which is responsible for scheduling the flows [30]. The real system attempts to emulate the virtual GPS by always scheduling the flows with the largest normalized lag. A similar approach based on virtual times is also adopted by the iterative Fair Scheduler (*iFS*) [31].

The other approach to achieving fairness is based on throttling, wherein a "jumping window" traffic policing mechanism ensures that each flow transmits at most a pre-computed fair share in each transmission window [32, 33]. A flow is throttled if it attempts to exceed its fair share and most of the algorithms differ in the calculation of the fair share. One approach has been to use the bucket filling algorithm in which the allocation of each of the flows is incremented by an infinitesimally small amount until either all available resources are exhausted or the flow's demands are met [8, 33]. The Fair-Maximum Weight Matching (F-MWM) algorithm proposed in [34] is based on an interesting duality between the Longest Queue First scheduling policy and max-min fairness. It was proposed for the inadmissible traffic case assuming that an input port is never the bottleneck link. It again adopts a throttling approach to implement fairness.

None of the afore-mentioned algorithms are based on a rigorous framework based on the notion of max-min fairness applied to input-queued switches. Algorithms based on emulating GPS incur the overhead of either maintaining

the virtual times, or emulating a virtual system or both. Algorithms based on the throttling inhibit performance by not permitting a flow to exceed its pre-allocated share even if there is no other flow contending for access. This is especially an issue in a network that only provides best-effort service and where flow demands change frequently.

4 Extending Max-Min to an Input-Queued Switch

While max-min fair allocation rates are easily computed in the case of a single shared resource, it is non-trivial to compute the max-min fair rates in the case of multiple flows competing for different sets of resources as in an input-queued switch. In this section we propose an ideal, iterative and max-min fair scheduler called the Fair Resource Allocation algorithm (FRA) for input-queued switches. Given a traffic pattern and the demands of all the flows, FRA computes the max-min fair rates for each of the flows.

4.1 Notation and System Model

In the following, given a matrix \mathcal{X} , we denote its element in the i -th row and j -th column by $\mathcal{X}[i, j]$. The i -th row vector and the j -th column vector of the matrix are denoted by $\mathcal{X}[i, *]$ and $\mathcal{X}[*, j]$, respectively. We denote the k -th element of a row vector \mathcal{C} by $\mathcal{C}[k]$.

Consider an $N \times N$ input-queued switch with N virtual output queues at each input port. Let the matrix \mathcal{F} denote the set of all flows entering the switch, where the element $\mathcal{F}[u, v]$ denotes the flow which arrives at input port u and is headed to output port v . Let the matrix \mathcal{D} denote the bandwidth demands of the flows. Similarly, let the matrix \mathcal{A} denote the bandwidth allocations given to the flows. Let the N -element row vector \mathcal{I} denote the peak bandwidth of the input links into the switch, where the u -th element denotes the peak bandwidth available at input port u . Similarly, let the N -element column vector \mathcal{O} denote the peak bandwidth at the output links of the switch, where the v -th element denotes the peak bandwidth available at output port v .

Given a scalar constraint such as the total amount of the resource available, the max-min fair allocation operator, $\text{MMF}()$, defined in Section 2.1, operates on a vector of demands to generate a vector of allocations. We now

define two new operators that operate on a matrix (as opposed to a vector) of demands. The *row-wise* max-min fair allocation operator, $\text{MMF}_R(\mathcal{S}, \mathcal{X})$, generates a matrix of allocations where each row of the allocation matrix is max-min fair with the demand vector being given by the corresponding row of the matrix \mathcal{X} and the constraint being given by the corresponding element of the vector of constraints \mathcal{S} . Similarly, the *column-wise* max-min fair allocation operator, $\text{MMF}_C(\mathcal{S}, \mathcal{X})$, generates a matrix of allocations where each column is max-min fair with the demand vector being given by the corresponding column of the matrix \mathcal{X} and the constraint being given by the corresponding element of the vector \mathcal{S} . In other words, the i -th row of the matrix generated by $\text{MMF}_R(\mathcal{S}, \mathcal{X})$ is $\text{MMF}(\mathcal{S}[i], \mathcal{X}[i, *])$. Similarly, the j -th column of the matrix generated by $\text{MMF}_C(\mathcal{S}, \mathcal{X})$ is $\text{MMF}(\mathcal{S}[j], \mathcal{X}[* , j])$.

Since MMF_R and MMF_C are max-min fair allocators as per the definition in Section 2.1, and since they operate on a single shared resource (output port bandwidth or input port bandwidth) they inherit all the properties of the $\text{MMF}()$ operator, including the properties *P1–P3* discussed in Section 2.1.

4.2 Max-Min Fairness in Input-Queued Switches

This section proposes an extension of the notion of max-min fairness to input-queued switches. The extension is straight-forward since, for best-effort traffic, all flows have equal rights to all the ports.

Any allocation in an input-queued switch must first be feasible, where feasibility is defined as follows:

Definition 5 The allocation matrix \mathcal{A} is said to be feasible if and only if:

1. The allocation of each flow is greater than or equal to zero; i.e., for all u and v , $\mathcal{A}[u, v] \geq 0$.
2. The total allocated bandwidth at each input port is less than or equal to the available bandwidth at that input port; i.e., for all u

$$\sum_{\forall v} \mathcal{A}[u, v] \leq \mathcal{I}[u]$$

3. The total allocated bandwidth at each output port is less than or equal to the available bandwidth at that output port; That is, for all v ,

$$\sum_{\forall u} \mathcal{A}[u, v] \leq \mathcal{O}[v]$$

Given a feasible allocation in an input-queued switch, we can establish whether it is max-min fair by verifying if increasing the allocation of any flow at any given port necessarily results in decreasing the allocation of some other flow with a smaller allocation at any port. This leads to the following definition:

Definition 6 The allocation matrix \mathcal{A} is said to be max-min fair if and only if:

1. It is feasible as per definition 5.
2. No entity receives an allocation greater than its demand; i.e., for all u and v ,

$$\mathcal{A}[u, v] \leq \mathcal{D}[u, v]$$

3. For all u and v , the allocation $\mathcal{A}[u, v]$ of a flow $\mathcal{F}[u, v]$ cannot be increased while satisfying the above two conditions and without reducing the allocation $\mathcal{A}[u', v']$ of another flow $\mathcal{F}[u', v']$, where $\mathcal{A}[u', v'] \leq \mathcal{A}[u, v]$.

4.3 The FRA Algorithm

In this section, we describe the Fair Resource Allocation (FRA) algorithm, which computes a max-min fair allocation in an input-queued switch. This section also provides its pseudo-code and illustrates its operation.

The operation of the algorithm can be divided into two main phases. In the first phase, all the output ports which are bottleneck links for at least one flow are allocated, thus finalizing the allocations for such flows. The demand matrix is updated to reflect these allocations and the process is repeated until there is no output port which is the bottleneck link for any flow. Thus, at the end of the first phase, the only possible bottlenecks for the flows are the input ports. In the second phase, the algorithm allocates bandwidth at the input ports in a max-min fair fashion, thus resulting in an allocation that is overall max-min fair.

The pseudo-code for the algorithm is presented in Fig. 2. The operation of the algorithm in the first phase (lines 02–12) is comprised of multiple iterations with each iteration consisting of three steps. In the first step (line 04), a tentative allocation of bandwidth for all the flows is computed considering only the constraints at the input ports (\mathcal{I}) by using the row-wise max-min fair operator MMF_R . This allocation (\mathcal{R}) is the maximum

```

01: Function FRA ( $\mathcal{D}, \mathcal{I}, \mathcal{O}$ ) returns  $[\mathcal{A}]$ 
02: Do
03:   done  $\leftarrow$  true
04:    $\mathcal{R} \leftarrow \text{MMF}_R(\mathcal{I}, \mathcal{D})$ 
05:    $\mathcal{G} \leftarrow \text{MMF}_C(\mathcal{O}, \mathcal{R})$ 
06:   for all  $[i, j]$ 
07:     if  $\mathcal{G}[i, j] < \mathcal{R}[i, j]$ 
08:        $\mathcal{D}[i, j] \leftarrow \mathcal{G}[i, j]$ 
09:       done  $\leftarrow$  false
10:     end if
11:   end for
12: While done is false
13:    $\mathcal{R} \leftarrow \text{MMF}_R(\mathcal{I}, \mathcal{D})$ 
14:   for all  $[i, j]$ 
15:     if  $\mathcal{R}[i, j] < \mathcal{D}[i, j]$ 
16:        $\mathcal{D}[i, j] \leftarrow \mathcal{R}[i, j]$ 
17:     end if
18:   end for
19:    $\mathcal{A} \leftarrow \mathcal{D}$ 
20: return  $\mathcal{A}$ 

```

Figure 2: Pseudo-code for the Fair Resource Allocation (FRA) algorithm.

possible allocation a flow can receive without considering the constraints at the output ports. While this ensures that the allocation is feasible at each of the input ports, it may not be feasible at the output ports. Hence, the next second step (line 05), computes the tentative fair allocation for all the flows by applying the constraints at the output ports (\mathcal{O}) on the allocations (\mathcal{R}) calculated in the previous line. This results in an allocation, \mathcal{G} , that is feasible at both the input and the output ports. Furthermore, since the above two steps use the row-wise and column-wise max-min fair operators, the allocations (\mathcal{G}) computed in the second step, are guaranteed to be less than or equal to the allocations (\mathcal{R}) computed in the first step. If the allocation of a flow computed in the second step (line 05) is less than the allocation in the first step (line 04), then the output port forms the bottleneck

link for that flow. In such a case, the flow’s allocation is considered *final* and it will not be allocated any additional resources. In order to take this flow out of subsequent iterations, the algorithm sets the demand of the flow to this computed allocation since no flow is ever allocated more than its demand in max-min operations (line 08). On the other hand, if the allocations computed for a flow in the above two steps are equal, then the bottleneck link cannot be identified and the allocation for the flow may be increased. In this case, the flow’s allocation is not yet considered final, and is allowed to participate in subsequent iterations (lines 02–12). The first phase of the algorithm ends when no new output port is found which is the bottleneck link for a flow whose allocation has not yet been finalized. The second phase of the algorithm (lines 13–20) completes the allocation by considering only the constraints at the input ports. If the calculated allocation is less than the demand of a flow, it implies that the input port is the bottleneck link and the demand of that flow is reduced accordingly (line 16). Otherwise, the demand of the flow can be satisfied (since neither an input port nor an output port forms its bottleneck link) and hence its allocation is set to its demand (line 20). Only one iteration of the second phase is required as either the input ports form the bottleneck links for the remaining unsatisfied flows, or the allocations at the input ports are able to satisfy the flows.

Fig. 3 shows the operation of FRA for the traffic pattern depicted in Fig. 3(a). We assume that all the flows are best effort and, therefore, their demands are equal to the peak line rate of the ports. Without loss of generality, we normalize the demands and the peak line rates of the ports and set them all to unity, which results in the initial demand vector shown in Fig. 3(b). The first execution of line 04 calculates the max-min fair allocation row-wise and, hence, results in the matrix depicted in Fig. 3(c). Column-wise fair allocation is then performed on this request matrix in line 05 to result in the matrix shown in Fig. 3(d). Since the tentative allocations calculated in the second step for flows AW, BW, BX and CW are smaller than the those calculated in the first step, lines 06–11 finalize their demands to result in the new demand matrix depicted in Fig. 3(f). The effect of this can be seen in the second iteration. Since flow CW is allocated one-fourth of the bandwidth available, flows CX and CY can now request a larger fraction of the bandwidth than in the previous iteration. This results in a corresponding decrease in the allocation to flow BX as shown in Fig. 3(h). Figs. 3(j)–3(l) show the operation of FRA in the third iteration of the *Do-While* loop. In the third iteration, the allocations calculated for all the flows are equal in

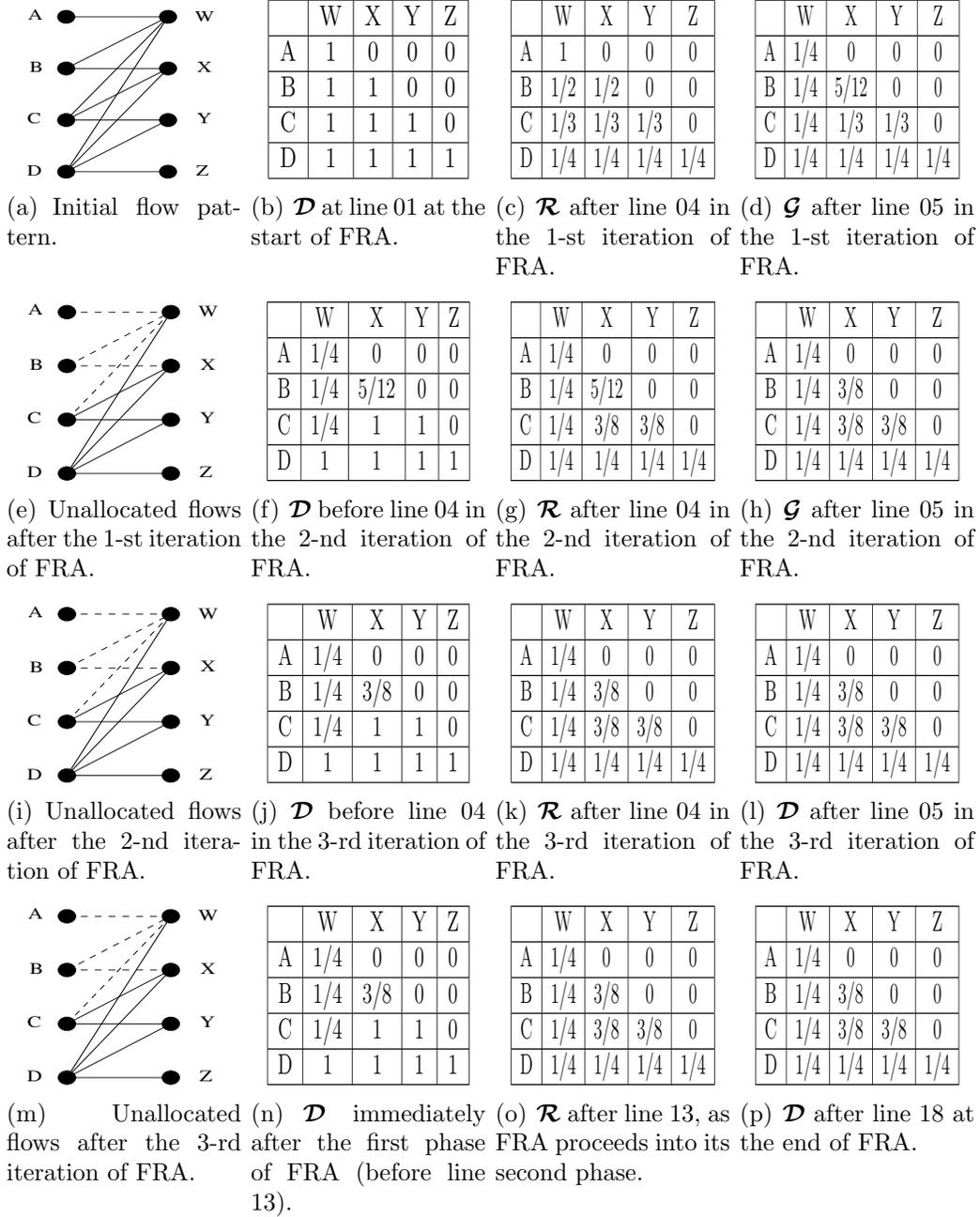


Figure 3: An illustration of the operation of FRA for a 4-port input-queued switch.

both the steps, and hence lines 06–11 never execute. Thus at the end of this iteration, the boolean variable *done*, which was previously set to true at line 03, remains true. The algorithm breaks out of the *Do-While* loop at line 12 and enters the second phase. The allocations for the flows is recalculated in line 13 using MMF_R operator applying the constraints of the input port resulting in Fig. 3(o). Lines 14–18 then compare the allocations of the flows CX, CY, DW, DX, DY and DZ to their demands. Since the allocations computed in line 13 are smaller than the demands of the flows (compare Figs. 3(n) and 3(o)), the input ports are the bottleneck links and the flows demands are reduced, resulting in the final allocation matrix shown in Fig. 3(p).

4.4 Correctness of the FRA Algorithm

In this section, we prove that the FRA algorithm results in a max-min fair allocation. We first identify certain properties of FRA that identify the conditions due to which a flow may be considered satisfied or a port may be considered fully allocated.

Lemma 1 A flow $\mathcal{F}[u, v]$ is satisfied if, during a complete execution of the FRA algorithm, its demand $\mathcal{D}[u, v]$ is not reduced by the algorithm.

Proof: If the flow’s demand is not modified by the algorithm, its final allocation is equal to its original demand as determined by line 20. Its demand having been met, the flow is, therefore, satisfied. ■

Definition 7 Upon completion of the execution of the FRA algorithm, an input port or an output port is said to have been *fully allocated* if the sum of the final allocations of all the flows passing through the port is equal to the capacity of the port.

Lemma 2 Upon the completion of the execution of the FRA algorithm, if there exists an unsatisfied flow $\mathcal{F}[u, v]$, then at least one of the ports, u or v , is fully allocated.

Proof: Consider the unsatisfied flow $\mathcal{F}[u, v]$ after the execution of the FRA algorithm. Since the flow is unsatisfied, we know from Lemma 1 that its demand is modified during the execution of the algorithm to a smaller value than the original demand. This modification occurs either at line 08, due to the condition $\mathcal{G}[u, v] < \mathcal{R}[u, v]$ (line 07), or at line 16 due to the condition

$\mathcal{R}[u, v] < \mathcal{D}[u, v]$ (line 15). In the first case, if the allocation calculated by the MMF_C operator at line 05, is smaller than that calculated by the MMF_R at line 04, we know that the output port v is fully allocated. In the other case, if the allocation calculated by the MMF_R operator on line 16 is less than the original demand of the flow, we know that the input port u is fully allocated, thus proving the lemma. ■

Note that the above lemma can be interpreted as a corollary of the proposition in [8] which states that a feasible rate vector is max-min fair if and only if each flow has a bottleneck link.

Lemma 3 Upon completion of the execution of FRA, if an unsatisfied flow $\mathcal{F}[u, v]$ exists, and input port u is not fully allocated, then none of the flows destined to output port v have an allocation greater than flow $\mathcal{F}[u, v]$; i.e., $\nexists \mathcal{F}[j, v]$, such that $l \neq u$ and $\mathcal{A}[j, v] > \mathcal{F}[u, v]$.

Proof: Since the flow is unsatisfied and the input port is not fully allocated, Lemma 2 implies that output port v was fully allocated. The properties P1–P3 of any max-min fair allocation policy, described in 2.1, further imply that there cannot be another flow terminating at output port v with an allocation greater than that of flow $\mathcal{F}[u, v]$. ■

Lemma 4 During the execution of FRA, if an unsatisfied flow $\mathcal{F}[u, v]$ exists, and output port v is not fully allocated, then none of the flows originating from input port u have an allocation greater than flow $\mathcal{F}[u, v]$; i.e., $\nexists \mathcal{F}[u, l]$, where $l \neq v$ such that $\mathcal{A}[u, l] > \mathcal{F}[u, v]$.

Proof: The proof is similar to that of Lemma 3.

Lemma 5 Given the allocations \mathcal{A} of the FRA algorithm, it is not possible to further increase the allocation of an unsatisfied flow without decreasing the allocation of some other flow with a smaller or equal allocation.

Proof: Let $\mathcal{F}[u, v]$ be an unsatisfied flow. From Lemma 2, we know that at least one of the ports, u and v , is fully allocated. We now consider three cases, depending on whether only the input port, only the output port, or both the ports are fully allocated by the algorithm.

Case 1 (input port u is not fully allocated): Since the flow is unsatisfied and the input port is not fully allocated, Lemma 2 implies that output port v is fully allocated. Lemma 3 further implies that $\mathcal{F}[u, v]$ has the largest allocation amongst all the flows destined to output port v . Thus, increasing the allocation of flow $\mathcal{F}[u, v]$ will necessarily decrease the allocation of another flow with a smaller or equal allocation at output port v .

Case 2 (output port v is not fully allocated): The proof is similar to that of the previous case.

Case 3 (both ports u and v are fully allocated): If there are no other flows which pass through u or v and which have an allocation larger than $\mathcal{A}[u, v]$, then the statement of the lemma is proved since, increasing the allocation of $\mathcal{F}[u, v]$ will necessarily result in decreasing the allocation of some other flow with a smaller or equal allocation. Consider the case where there is at least one flow that passes through either u or v and has a larger allocation. Without loss of generality, let us assume that this flow with the larger allocation passes through input port u . We will denote this flow by $\mathcal{F}[u, w]$, where $\mathcal{A}[u, w] > \mathcal{A}[u, v]$. Since the flow $\mathcal{F}[u, v]$ is not satisfied, and there are other flows with a larger allocation passing through u , the allocation of this flow is not constrained by the MMF_R operator which considers only the constraint at input port u . Therefore, the constraint on the allocation of flow $\mathcal{F}[u, v]$ is arising from the MMF_C operator at the output port v . Thus, at this output port, given the properties of any max-min fair allocation policy described in Section 2.1, there cannot be another flow terminating at output port v with an allocation greater than that of flow $\mathcal{F}[u, v]$. Thus, increasing the allocation of flow $\mathcal{F}[u, v]$ will necessarily decrease the allocation of another flow with a smaller or equal allocation at the output port v . ■

Theorem 1 FRA results in a max-min fair allocation.

Proof: Since the FRA allocations are through the max-min operators MMF_R and MMF_C , it satisfies the feasibility condition and ensures that no flow will receive an allocation larger than its original demand. The third condition is satisfied from Lemma 5.

5 Practical Allocation Strategies

In this section, we investigate how FRA might be implemented in input-queued switches. We propose a parallelized version of FRA, called Distributed Fair Resource Allocation (DFRA), which can easily be implemented in input-queued switches.

Given that real traffic is not infinitesimally divisible, scheduling as well as transmission on the output link must be done in the form of packets. To support this, DFRA allocates bandwidth in the form of credits and each flow consumes a credit to transmit a packet.

5.1 The DFRA Implementation

DFRA requires three temporary matrices: the demand matrix \mathcal{D} , the request matrix \mathcal{R} , and the grant matrix \mathcal{G} , in addition to two vectors: the input bandwidth vector \mathcal{I} , and the output bandwidth vector \mathcal{O} . All of the above are maintained in a distributed fashion.

The demand matrix \mathcal{D} , the request matrix \mathcal{R} , the input constraint vector \mathcal{I} and the output constraint vector \mathcal{O} are maintained in a distributed fashion. Each row of the demand matrix is maintained by an input port; i.e., input port i maintains $\mathcal{D}[i, *]$ with each element being associated with a virtual output queue. Input port i also maintains the free bandwidth available $\mathcal{I}[i]$, i.e., the i -th element of the input constraint vector. All the input ports generate each row of the request vector independently and in parallel, given by,

$$\mathcal{R}[i, *] = \text{MMF}(\mathcal{I}[i], \mathcal{R}[i, *]).$$

After the request matrix has been generated, the transpose of the request matrix is delivered to the output ports. More specifically, the j -th row of \mathcal{R}^T is transferred to output port j . That is, the j -th column of \mathcal{R} is delivered to the j -th output port. In practice, each input port notifies every output port of its tentative allocation, thus generating the request vector to all the output ports. Once the output ports receive \mathcal{R}^T , they perform the max-min fair allocation independently and in parallel, given by

$$\mathcal{G}[* , j] = \text{MMF}(\mathcal{O}[j], \mathcal{R}[* , j])$$

The grant matrix is then delivered row-wise to the input ports; i.e., $\mathcal{G}[i, *]$ is delivered to input port i . Each input port then compares the grant received by a flow to the request issued for that flow. If the grant is smaller than the request, then the output port for that flow is more congested than the input port and hence the possibility of increasing its allocation does not exist. The flow is therefore considered satisfied and its demand is set to its grant.

Thus, each iteration of DFRA can be run in parallel. The elements of the request and grant vectors can be piggy-backed on top of the requests, grants and accepts issued in iterative matching algorithms and hence do not entail any additional communication complexity. The remaining issue is the number of iterations DFRA needs to complete. The next section proves that DFRA requires at most N iterations. However, we shall present simulation results showing that a smaller number of iterations are sufficient in practice.

5.2 Iterations Required

The time complexity of DFRA is determined primarily by the maximum number of times DFRA executes the statements in the *Do-While* loop (lines 02–12) since the number of operations executed after the loop (lines 13–20) is a constant. In this section, we show that DFRA requires at most N iterations of the *Do-While* loop to completely allocate all available bandwidth in a max-min fair fashion to all the contending flows. The condition DFRA must encounter to exit the *Do-While* loop is stated by the following lemma.

Lemma 6 DFRA exits the *Do-While* loop (lines 02–12) if it does not modify the demand of any flow during an iteration of the loop.

Proof: If DFRA does not modify the demand of any flow, then the value of the boolean variable *done*, which was set to *true* in line 03, is not modified in line 09. Therefore DFRA breaks out of the *Do-While* loop at line 12. ■

Given the above condition, we now have to identify the largest number of iterations which may occur before DFRA encounters the above condition.

Lemma 7 For DFRA to continue with another iteration of the *Do-While* loop (lines 02–12), at least one output port must be completely allocated in the current iteration.

Proof: The proof of this lemma can be derived as an extension of the previous lemma. For the algorithm to proceed to another iteration, Lemma 6 must be false. If Lemma 6 is false, then the only reason a flow has its demand modified is because the grant was less than its request (line 07). The MMF_C operator used by the output ports ensures that this happens only when the output port is completely allocated and has no available bandwidth. This proves the lemma. ■

Theorem 2 The DFRA algorithm takes N iterations to allocate bandwidth in a max-min fair fashion in an input-queued switch.

Proof: From Lemma 7, each iteration leads to at least one port being fully allocated. Hence for a switch with N output ports, DFRA requires N iterations. ■

While the DFRA algorithm takes N iterations in the worst case to complete the allocations, significant improvements in fairness can be achieved by running DFRA for only a small number of iterations. We demonstrate this through simulation using real traffic traces in the next section.

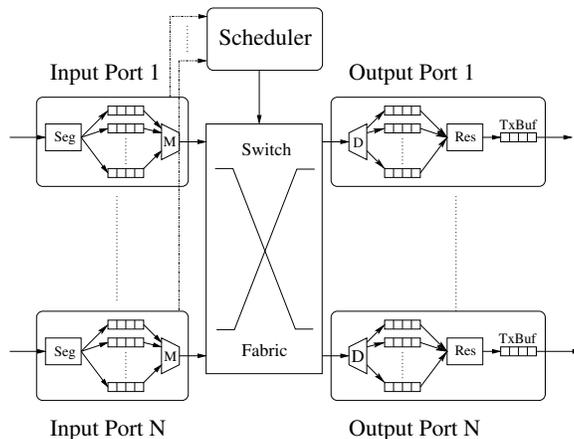


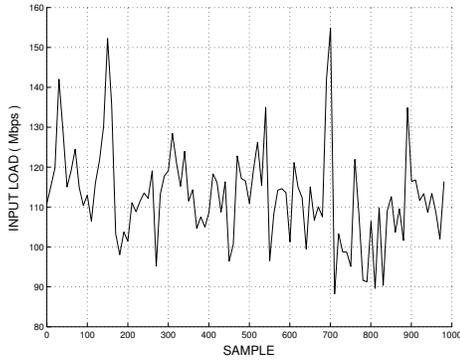
Figure 4: Architecture of the input-queued switch used for simulation.

6 Simulation Results and Analysis

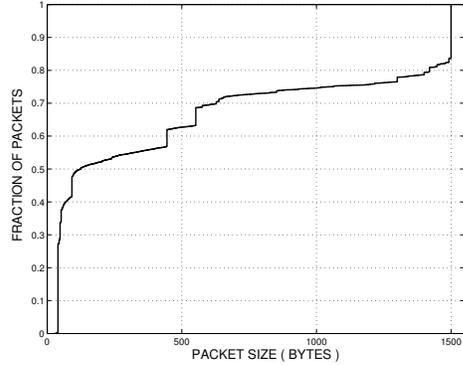
6.1 Simulation Model

Our simulation experiments assume an N -input, N -output input queued switch that uses virtual output queueing, as shown in Fig.4. IP packets arriving at each input port are first broken down into fixed-size cells of 48 bytes by the segmentation (*Seg*) module, after which they are placed into the virtual output queue (VOQ) for their destination port. The destination output port of each packet is determined by taking the sum (modulo the number of output ports) of individual bytes of the destination IP address of the packet; i.e., given a destination IP address $a.b.c.d$, the destination output port is determined by $(a+b+c+d)\bmod N$. This ensures that packets destined to the same destination IP address contend for access to the same output port.

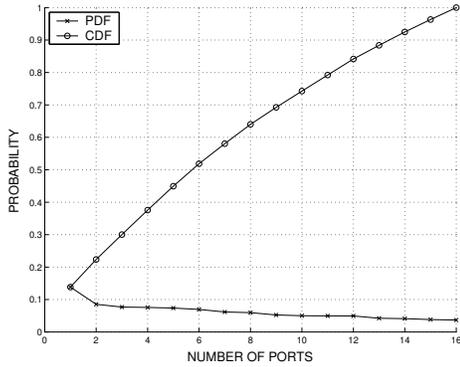
Each input port keeps the *Scheduler* updated of the state of its virtual output queues and the *Scheduler* is responsible for configuring the crossbar for each transmission slot. Cells arriving at an output port are first stored in the reassembly buffers. When the last cell of a packet arrives into the reassembly buffers, the reassembly module (*Res*) extracts all the cells, reassembles them into the complete IP packet and enqueues the packet into the transmission queue (*TxBuf*). The transmitter, which is not shown in this figure, dequeues



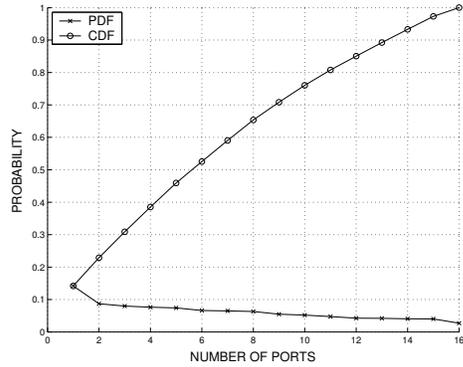
(a) Instantaneous input load averaged over 100 clock ticks.



(b) CDF of the packet sizes.



(c) Representative destination port distribution.



(d) Representative source port distribution.

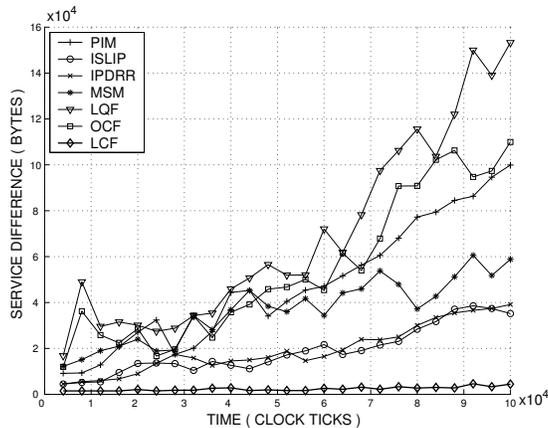
Figure 5: Characteristics of the real traffic traces obtained from NLANR.

packets from this queue and sends them out on the link. M and D are multiplexors and demultiplexors, respectively, which control access to the queues. Sufficient control logic is assumed to be present at the ports to manage the multiplexors and demultiplexors and other control signals.

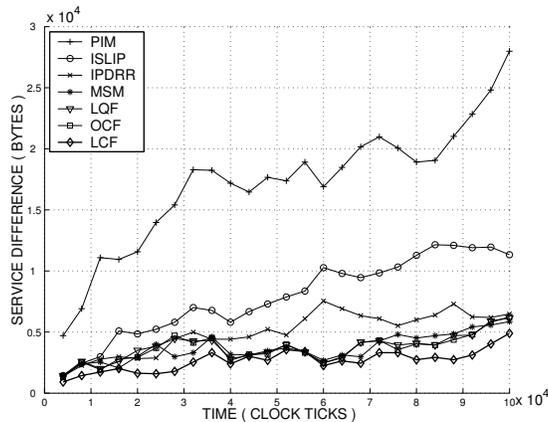
Simulations were performed for a 16-port switch operating with a per-port bandwidth of 155 Mbps using seven different scheduling algorithms listed below:

- PIM, for its historical significance

- *i*SLIP, also for its historical significance
- IPDRR, for its ability to provide guaranteed QoS and its superior performance over the iterative Fair Scheduler (*i*FS) [18, 31]. To ensure maximum fairness and since all the flows were best-effort traffic, the quota of each flow is set to 1.
- The maximum size matching (MSM) algorithm, for its promise of maximum throughput.
- A variant of maximum weight matching, called the Longest Queue First (LQF) where the edge weights are equal to the number of cells in the corresponding VOQ [20].
- Another variant of maximum weight matching called the Oldest Cell First (OCF) algorithm, where the edge weights are equal to the queuing delay of the head-of-line packet in the corresponding VOQ [20].
- A new variant of maximum weight matching algorithm that we propose, where the weight of an edge is the number of credits accumulated by the flow. A flow is considered backlogged if it has at least one packet enqueued at the virtual output queues and is considered inactive for that cycle if it does not have any packets queued. At each cycle, the demand of a backlogged flow is set to one and that of an inactive flow is set to zero. DFRA is then used by the switch to calculate the max-min fair allocation, and credits equal to the allocation are issued to the flow. These allocated credits are accumulated by the flows and a flow utilizes a credit whenever it transfers a packet from the input ports to the output ports. We call this variant the *Largest Credit First (LCF)* algorithm. LCF permits flows to transmit even if they do not have credits for that particular cell. Hence, the credits accumulated by a flow may be negative. This strategy ensures that a flow with negative credits will receive less preference than a flow with positive credits in the maximum weight matching algorithm, but may still be matched if there is no other flow with a larger number of credits that may be matched.
- Finally, the ideal FRA algorithm to serve as a benchmark against which all other algorithms can be measured.



(a) Using real traffic traces.



(b) Using synthetic traffic.

Figure 6: Maximum of the difference in the service received by a flow under a given scheduling strategy and the service received under FRA, measured in terms of the number of bytes transmitted.

6.2 Traffic Characteristics

Simulations were performed using both real and synthetic traffic traces. Real traffic traces were obtained from the Passive Measurement and Analysis (PMA) web-site of the National Laboratory for Applied Network Research (NLNR) [35]. The traces selected were from the Colorado State University site, which taps an *OC3c* link using a pair of network cards synchronized by

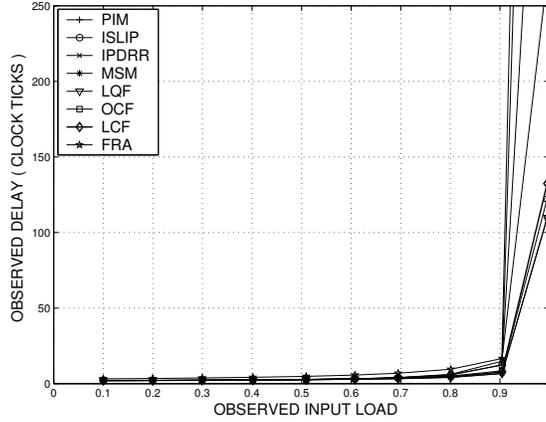
cable [36]. The original trace file contained traces of about 2.5×10^6 packets, representing about 90 seconds of network activity.

Fig. 5 describes some characteristics of the traffic traces. The average input traffic rate was observed to be around 112 Mbps per port although the instantaneous load varied significantly from 90 Mbps to 155 Mbps. Fig. 5(a) plots the average input load observed during each interval of 100 clock ticks. The size of the packets varied from 64 bytes to 1500 bytes with a significant fraction of packets being either 64 or 1500 bytes long. Fig. 5(b) shows the cumulative distribution function of the sizes of packets. Further, the traffic was also observed to be non-uniform, with a single output port being the destination for 15% of the packets arriving at the input port, while another output port is the destination for only 5% of the packets arriving at the same input port, as shown by Fig. 5(c). This non-uniformity of the traffic is also evident in Fig. 5(d) which plots the distribution of the sources of packets arriving at a representative output port. Almost 15% of the packets arriving at an output port originate from a single input port.

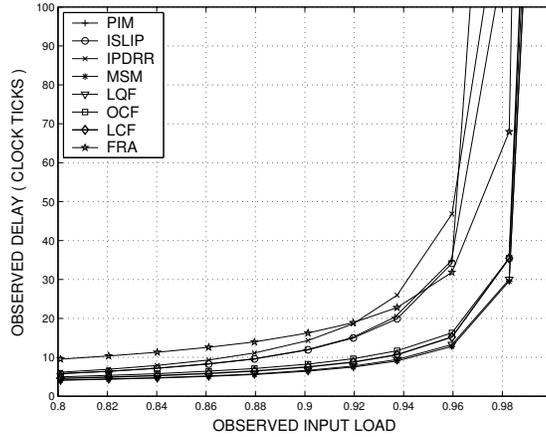
In order to analyze the performance characteristics, simulations were also performed using synthetic traffic. Poisson traffic at fractions varying from 0.1 to 1.0 of the line rate were generated with the packet destinations being uniformly distributed. Packet sizes were assumed to be a constant 48 bytes. Hence, for simulations performed using synthetic traffic, the switch was assumed to operate with an internal per port bandwidth of $48 \times 8 = 384$ Mbps with no speedup.

6.3 Evaluation of Existing Schedulers

Fig. 6(a) plots the maximum difference in service received by a flow under different schedulers as compared to the service received by the flow under FRA. The LQF and OCF schedulers result in the largest difference as they tend to favor flows with the longest queues and largest delays, respectively. Thus, a flow which sends large bursts is typically given preference over flows with fewer bursts, even though, they should ideally receive the same service. PIM also performs poorly because its randomized approach is unable to guarantee fairness. Maximum size matching maximizes the overall throughput and is therefore more fair than LQF, OCF and PIM. *i*SLIP and IPDRR perform very well as they use a round robin approach. Once the output grant pointers and input accept pointers are desynchronized, *i*SLIP operates in a round robin fashion and is therefore more fair. IPDRR also implements a



(a) Across all input loads.



(b) Across input loads of interest

Figure 7: Performance comparison of FRA and LCF with other schedulers under synthetic uniform traffic.

strategy similar to *i*SLIP using linked lists instead of pointers and hence operates for the most part in a round robin fashion. Thus, the fairness of *i*SLIP and IPDRR is similar. This is corroborated by the results in [18] wherein the performance of both the algorithms is similar, though IPDRR supports guaranteed services while *i*SLIP does not. LCF is the most fair as it closely emulates FRA. However, LCF cannot perfectly match the fairness of FRA because LCF clears the credits accumulated by a flow when it is no longer

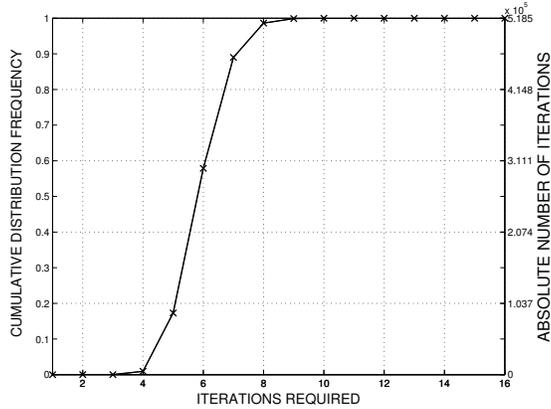
backlogged. This loss of information about the service received by a flow when it was previously backlogged results in the slight increase in unfairness. With continuously backlogged traffic, however, the LCF algorithm will approach the ideally fair rates as computed by the FRA algorithm.

Fig. 6(b) plots the same statistic using synthetic traffic, generated as described previously. Since packet destinations are uniformly distributed amongst all the output ports, the maximum difference between the service received under these schedulers and FRA closely follows the throughput achieved under these schedulers, with MSM, LQF and OCF having superior fairness over IPDRR and *i*SLIP. LCF remains the most fair algorithm.

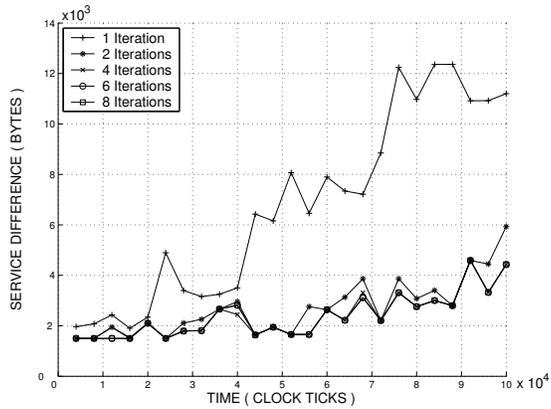
Fig. 7(a) plots the performance of these scheduling algorithms under synthetic traffic with Fig. 7(b) focussing on the region of interest. The results corroborate previous results studying the performance of these schedulers reported in [32] and [33]. In addition, it can be seen that the performance difference of LCF as compared to other high-performance schedulers such as LQF, OCF and MSM is negligible. FRA, although not implementable, exhibits higher delay since it is not work-conserving; i.e., a packet that has not accumulated a credit will not be transmitted even if its destination output port is free.

6.4 Iterations Required

In this section, we study the impact of the number of iterations executed by LCF. Fig 8(a) shows the number of iterations required by LCF in the simulations using real traffic traces. In approximately 60% of the instances, LCF requires only 6 iterations with 8 iterations being sufficient in over 99% of the instances. The number of iterations can in fact be reduced even further since the law of diminishing returns holds; i.e., the first iteration will typically result in the largest reduction of demands with succeeding iterations resulting in smaller reductions of the demands. This is illustrated by Fig 8(b), which shows the effect on fairness as the number of iterations performed for the LCF matching algorithm are increased from 2 to 8. Negligible improvement in fairness is achieved as the number of iterations is increased beyond two. In addition, LCF permits a flow to transmit even though it may have exceeded its fair allocation, as opposed to the algorithms in [33] and [18]. This LCF strategy with just two iterations achieves fairness very close to that of the DFRA algorithm with N iterations. In fact, our simulations indicate that the credits accumulated by a flow never exceed +1 or -1.



(a) Distribution of the number of iterations required by DFRA for real traffic traces.



(b) Impact on fairness.

Figure 8: Effect of number of iterations required by DFRA for real traffic traces.

7 Concluding Remarks

While the problem of allocating bandwidth *fairly* over a single shared link has been studied extensively in the research literature, overall fairness in a switch cannot be achieved by individually allocating each link fairly among the different flows. In this report, we extend the notion of max-min fairness defined for a single resource to the context of an input-queued switch. We propose a new algorithm, called the *Fair Resource Allocation (FRA)*, that

exactly computes the max-min fair rates for each flow through the network. A unique aspect of this algorithm is that it lends itself very easily to an iterative distributed implementation in an input-queued switch.

The rates computed by the FRA algorithm also serve as a reference in the evaluation of the fairness achieved by various schedulers for input-queued switches. Based on this method, we evaluate the fairness achieved by several scheduling algorithms for input-queued switches through simulation using real traffic traces as well as synthetically generated traffic. We propose a distributed implementation of the FRA which achieves better fairness than other known schedulers for input-queued switches, without incurring any significant loss in performance. We also suggest a simple variant of maximum weighted matching, called *Largest Credit First* where the weight is proportional to the credits allocated to a flow based on a distributed implementation of FRA. We show that the LCF algorithm with just two iterations is able to achieve both good fairness and performance.

References

- [1] S. Keshav, *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network*, 1st ed. Addison-Wesley, May 1997.
- [2] T. Bonald and L. Massouli, "Impact of fairness on internet performance," in *Proc. ACM SIGMETRICS*, vol. 29, no. 1. ACM Press, June 2001, pp. 82–91.
- [3] D. Cohen and K. Narayanaswamy, "A fair service approach to defenses against packet flooding attacks," White Paper. [Online]. Available: <http://www.cs3-inc.com/ddos.html>
- [4] J. Xu and W. Lee, "Sustaining availability of web services under distributed denial of service attacks," *IEEE Trans. Comput.*, vol. 52, no. 2, pp. 195–208, February 2003.
- [5] D. K. Y. Yau, J. C. S. Lui, F. Liang, and Y. Yam, "Defending against distributed denial of service attacks with max-min fair server-centric router throttles," *IEEE/ACM Trans. Networking*, vol. 13, no. 1, pp. 29–42, February 2005.

- [6] F. Kelly, “Charging and rate control for elastic traffic,” *European Trans. on Telecommunications*, vol. 8, no. 1, pp. 33–37, 1997.
- [7] Z. Cao and E. W. Zegura, “Utility max-min: An application-oriented bandwidth allocation scheme,” in *Proc. IEEE INFOCOM*, vol. 2, New York, NY, USA, March 1999, pp. 793–801.
- [8] D. Bertsekas and R. Gallager, *Data Networks*, 2nd ed. Prentice Hall, 1992.
- [9] B. Vandalore, S. Fahmy, R. Jain, R. Goyal, and M. Goyal, “General weighted fairness and its support in explicit rate switch algorithms,” *Computer Communications*, vol. 23, no. 2, pp. 149–161, January 2000.
- [10] D. Liu, N. Ansari, and E. Hou, “Fairness criterion for allocating resources in input queued switches,” *IEE Electronic Letters*, vol. 37, no. 19, pp. 1205–06, September 2001.
- [11] A. K. Parekh and R. G. Gallager, “A generalized processor sharing approach to flow control in integrated services networks: The single-node case.” *IEEE/ACM Trans. Networking*, vol. 1, no. 3, pp. 344–357, June 1993.
- [12] D. C. Stephens, J. C. R. Bennett, and H. Zhang, “Implementing scheduling algorithms in high-speed networks,” *IEEE J. Select. Areas Commun.*, vol. 17, no. 6, pp. 1145–1158, June 1999.
- [13] Cisco Systems Inc., “Cisco IOS software release 12.0(5)T,” 1999. [Online]. Available: <http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/120newft/120t/120t5/cbwfq.pdf>
- [14] J. M. Kleinberg, E. Tardos, and Y. Rabani, “Fairness in routing and load balancing,” in *Proc. 40-th Annual Symp. on Foundations of Computer Science (FOCS)*. New York, NY, USA: IEEE Computer Society, 1999, pp. 568–578.
- [15] L. Tassiulas and S. Sarkar, “Maxmin fair scheduling in wireless networks,” in *Proc. IEEE INFOCOM*, vol. 2, New York, NY, USA, June 2002, pp. 763–772.

- [16] S. Sarkar and K. N. Sivarajan, “Fairness in wireless mobile networks,” *IEEE Trans. Inform. Theory*, vol. 48, no. 8, pp. 2412–2426, August 2002.
- [17] A. Kam and K.-Y. Siu, “Linear complexity algorithms for QoS support in input-queued switches with no speedup,” *IEEE J. Select. Areas Commun.*, vol. 17, no. 6, pp. 1040–1056, June 1999.
- [18] X. Zhang and L. Bhuyan, “Deficit round robin scheduling for input queued switches,” *IEEE J. Select. Areas Commun. Special Issue on High Performance Optical/Electronic Switches/Routers for High Speed Internet*, vol. 21, no. 4, pp. 584–594, May 2003.
- [19] Y. Tamir and G. L. Frazier, “High-performance multi-queue buffers for vlsi communications switches,” in *Proc. 15-th Annual Intl. Symp. on Comp. Arch. (ISCA)*. Honolulu, Hawaii, USA: IEEE Computer Society Press, 1988, pp. 343–354.
- [20] 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, August 1999.
- [21] Hopcroft and Karp, “An $n^{\frac{5}{2}}$ algorithm for maximum matchings in bipartite graphs,” *SIAM J. on Comput.*, vol. 2, no. 4, p. 225, December 1973.
- [22] I. Keslassy, R. Zhang-Shen, and N. McKeown, “Maximum size matching is unstable for any packet switch,” *IEEE Commun. Lett.*, vol. 7, no. 10, pp. 496–498, October 2003.
- [23] A. Charny, P. Krishna, N. Patel, and R. Simcoe, “Algorithms for providing bandwidth and delay guarantees in input-buffered crossbars with speedup,” in *Proc. 6-th Intl. Wksp. on Quality-of-Service (IWQoS 98)*, Napa, CA, USA, May 1998, pp. 235–244.
- [24] E. Leonardi, F. Neri, and B. Yener, “Algorithms for virtual output queued switching,” in *Proc. IEEE Global Telecommunications Conf. (GLOBECOM-99)*, vol. 2, Rio de Janeiro, Brasil, December 1999, pp. 1203–1210.

- [25] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, “High-speed switch scheduling for local-area networks,” *ACM Trans. Comput. Syst.*, vol. 11, no. 4, pp. 319–352, November 1993.
- [26] N. McKeown, “SLIP: A scheduling algorithm for input-queued switches,” *IEEE/ACM Trans. Networking*, vol. 7, no. 2, pp. 188–201, April 1999.
- [27] D. C. Stephens and H. Zhang, “Implementing distributed packet fair queueing in a scalable switch architecture,” in *Proc. IEEE INFOCOM*, San Francisco, CA, USA, April 1998, pp. 282–290.
- [28] D. C. Stephens, “Implementing distributed packet fair queueing in a scalable switch architecture,” Master’s thesis, Carnegie Mellon University, 1998.
- [29] M. Shreedhar and G. Varghese, “Efficient fair queueing using deficit round-robin,” *IEEE/ACM Trans. Networking*, vol. 4, no. 3, pp. 275–285, June 1996.
- [30] S. Li, J.-G. Chen, and N. Ansari, “Fair queueing for input-buffered switches with back pressure,” in *Proc. 1-st IEEE Intl. Conf. on ATM (ICATM-98)*, Colmar, France, June 1998, pp. 252–259.
- [31] N. Ni and L. N. Bhuyan, “Fair scheduling in internet routers,” *IEEE Trans. Comput. Special Issue on Quality of Service Issues in Internet Web Services*, vol. 51, no. 6, pp. 686–701, June 2002.
- [32] D. Cavendish, M. Goudreau, and A. Ishii, “On the fairness of scheduling algorithms for input-queued switches,” in *Proc. 17-th Intl. Teletraffic Congress*, vol. 4, Brazil, December 2001, pp. 829–841.
- [33] D. Cavendish, M. Lajolo, and H. Liu, “On the evaluation of fairness for input queued switches,” in *Proc. IEEE Intl. Conf. on Communications (ICC-2002)*, vol. 2, May 2002, pp. 996–1000.
- [34] N. Kumar, R. Pan, and D. Shah, “Fair scheduling in input-queued switches under inadmissible traffic,” in *Proc. IEEE Global Telecommunications Conf. (GLOBECOM-04)*, vol. 3, no. 29, December 2004, pp. 1713–1717.

- [35] *Passive Measurement and Analysis*. NLANR. [Online]. Available: <http://pma.nlanr.net/PMA/>
- [36] *COS-1069192584-1.tsh.gz taken from the NLANR Colorado State University OC3c tap on Nov 18, 2003.*, This trace may be requested from the PMA HPSS repository. [Online]. Available: <http://pma.nlanr.net/PMA/Sites/COS.html>