Fair Scheduling in Networks Through Packet Election

Fair Scheduling in Networks Through Packet Election

Srikanth Jagabathula and Devavrat Shah

***Abstract*—We consider the problem of designing a fair scheduling algorithm for discrete-time constrained queuing networks. Each queue has dedicated exogenous packet arrivals. There are constraints on which queues can be served simultaneously. This model effectively describes important special instances like network switches, interference in wireless networks, bandwidth sharing for congestion control and traffic scheduling in road roundabouts. Fair scheduling is required because it provides isolation to different traffic flows; isolation makes the system more robust and enables providing quality of service. Existing work on fairness for constrained networks concentrates on flow based fairness. As a main result, we describe a notion of packet based fairness by establishing an analogy with the ranked election problem: packets are voters, schedules are candidates and each packet ranks the schedules based on its priorities. We then obtain a scheduling algorithm that achieves the described notion of fairness by drawing upon the seminal work of Goodman and Markowitz (1952). This yields the familiar Maximum Weight (MW) style algorithm. As another important result, we prove that the algorithm obtained is throughput optimal. There is no reason a priori why this should be true, and the proof requires non-traditional methods**

***Index Terms*—Fair scheduling, packet-based fairness, ranked election, throughput optimality**

I. Introduction

In this paper, we focus on the problem of scheduling in constrained queuing networks. Specifically, we consider a collection of queues operating in discrete time with constraints on which queues may be served simultaneously. Such queuing systems serve as effective modeling tools for a large array of important practical problems, and their performance is crucially dependent on the effectiveness of the scheduling algorithm.

In this setup, the basic question is to design a scheduling algorithm that is optimal. There are several performance criteria, with often inherent trade-offs, that determine the optimality of a scheduling algorithm. The first is throughput optimality. A queuing system has a limited amount of resources. The natural constraints imposed result in an inherent limitation on the amount of traffic load that can be supported. This is called the capacity of the system. Roughly speaking, a scheduling algorithm that achieves the capacity utilizes system resources optimally. Such an algorithm is called throughput optimal.

This work was supported in parts by NSF CAREER CNS 0546590 and NSF CCF 0728554.

The authors are with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 02139 USA e-mail: {jskanth, devavrat}@ mit.edu

Apart from being throughput optimal, a scheduling algorithm should allocate resources in a fair manner. The queuing system is a common resource shared by various traffic flows, and the scheduling algorithm should ensure that no flow is receiving more than its "fair" share of resources. It is important to realize that fairness in queuing systems is not only an intuitively desired goal but also one with an immense practical impact.

A very important consequence of fairness is isolation of various traffic flows. Throughput optimality is oblivious to the identities of the different flows. But, identities are important for the following two important reasons: (1) Being oblivious to flow identities, throughput optimal algorithms often favor flows with a high data rate. Therefore, a particular flow might ill-behave and flood the system with a high data rate maliciously resulting in the deterioration of service to other flows. Since the system is a shared resource, the algorithm should identify the rogue flow and limit the negative impact on well-behaved flows. (2) Secondly, isolation is important to provide performance guarantees, and thereby Quality of Service (see Keshav (1997) [1]), to various flows in the system. Designing a scheduling algorithm that is fair will overcome these issues. Other benefits of fairness include reducing burstiness of flows, eliminating bottlenecks and reducing the impact of certain kinds of Denial-of-Service (DoS) attacks (see Bonald and Massoulie´ (2001) [2], and Yau et al (2005) [3]). In essence, a fair scheduling algorithm makes the queuing system robust and less prone to manipulation by individuals.

A natural way to achieve isolation among flows, in order to provide protection and performance guarantees, is to dedicate resources to each of the flows. In fact, this is the approach taken in most of the work done on designing fair algorithms for input queued switches (details in Section I-B). This approach, though, is limited for the following reasons: Firstly, because of constraints, it is not straightforward to determine the amount of resources to be allocated to each flow in a queuing network. Moreover, such determination would require the knowledge of flow arrival rates; whereas, in the spirit of being implementable, we require the scheduling algorithm to be online i.e., use only current network state information like queue-sizes, age of packets, etc., and myopic i.e., oblivious to flow arrival rates. Secondly, resource allocation takes place on an average over a long period of time. This is appropriate in a flow level model where the arrival statistics remain constant for long time periods. This assumption, though, is questionable in many applications like Internet traffic where short flows predominate.

We note that designing a fair scheduling algorithm com-

prises two sub-problems: defining a reasonable notion of fairness, and designing an algorithm to achieve the desired notion of fairness. The notion of fairness is utilized to determine the resources (more specifically, the rate or bandwidth) to be allocated, and the scheduling algorithm ensures that the resources are allocated on an average over a long period of time. Inspired by this, our approach would be to first define a notion of packet based fairness and then design a scheduling algorithm to achieve the defined notion of fairness. For obvious reasons, we also need to reconcile the benefits of fairness with achieving throughput optimality.

Motivated by the above discussion, we attempt to achieve the following three goals in this paper: (1) Define a notion of packet based fairness. (2) Design an online and myopic algorithm, that is also throughput optimal, to achieve the notion of fairness. (3) Provide the throughput optimality proof of the algorithm.

A. Our contributions

The need for a packet based notion of fairness that can be used in a constrained network is clear. But, defining a precise mathematical notion of fairness that achieves the desired intuitive and practical benefits of fairness is quite challenging. Unfortunately, none of the existing notions of fairness directly extend to a packet based constrained queuing network. Existing notions of flow based fairness are based on the utility maximization framework (proposed by Kelly, Maullo and Tan (1998) [4]), which is a concept borrowed from Economics literature. In a similar spirit, we define a notion of fairness by establishing a novel analogy between scheduling in constrained queuing networks and a ranked election problem[1]. Ranked election is a widely studied problem in the Economics (and Political Science) literature and this analogy provides a ready framework to leverage this work. We draw upon the work of Goodman and Markowitz (1952) [5] to obtain a unique characterization of the schedule. This, rather surprisingly, yields a maximum weight (MW) style algorithm. MW style algorithms are very popular in the literature and are very well understood. Thus, MW algorithms choose schedules in a "fair" manner, though the definition of "fair" for different weights is different. It should be noted that the choice of weights is crucial for obtaining the intuitive desirable properties of fairness, and we make an important contribution here.

As another important contribution, we prove that our algorithm is throughput optimal. There is no a priori reason for this to be true. Even though the algorithm we design is the familiar MW style algorithm, it is not queue size or waiting time based. Therefore traditional methods of proving throughput optimality, which include the popular Lyapunov-Foster method and Fluid models, cannot be applied in a direct manner. The proof technique we introduce to prove throughput optimality is potentially applicable to a wider class of problems.

[1]A ranked election problem deals with choosing a winning permutation of candidates using a set of votes, where each vote is a permutation of the candidates. Refer to Section IV.

B. Related work

We first begin with the work on single queue fairness. Fair scheduling in single queues has been widely studied since the early 1990s. In one of the earliest works, John Nagle (1987) [6] proposed a fair algorithm for single queues called "Fair Queuing." As mentioned earlier, fair scheduling is required to minimize starvation and limit the negative impact of rogue sources. In order to achieve this, Nagle proposed maintaining separate queues for different flows and serving them in a round-robin fashion. This is a great and simple to implement solution, but it works only when all the packets are of equal size. In order to overcome this problem Demers, Keshav and Shenker (1989) [7] proposed the notion of Weighted Fair Queueing (WFQ) and its packetized implementation. Parekh and Gallager (1993, 1994) [8], [9] analyzed the performance of this packetized implementation and showed it to be a good approximation of Generalized Processor Sharing (GPS). Shreedhar and Varghese (1996) [10] designed a computationally efficient version of weighted fair queuing called Deficit Weighted Round Robin (DWRR). Even though all these algorithms are fair, they are very complex and expensive to implement. Hence, there was a lot of work done on achieving approximate fairness for Internet routers through FIFO queuing and appropriate packet dropping mechanisms. Examples include RED by Floyd and Jacobson (1993) [11], CHoKe by Pan, Prabhakar and Psounis (2000) [12], and AFD by Pan, Prabhakar, Breslau and Shenker (2003) [13].

To address the issue of fairness in a network, Kelly, Maullo and Tan (1998) [4] proposed a flow-level model for the Internet. Under this model, the resource allocation that maximizes the global network utility provides a notion of fair rate allocation. We refer an interested reader to survey-style papers by Low (2003) [14] and Chuang et.al. (2006) [15] and the book by Srikant (2004) [16] for further details. We take a note of desirable throughput property of the dynamic flow-level resource allocation model (see for example, Bonald and Massoulie´ (2001) [2], and de Veciana, Konstantopoulos and Lee (2001) [17]). This approach, though valid for a general network with arbitrary topology, does not take scheduling constraints into account.

We next review the work done on the design of fair scheduling algorithms for Input Queued (IQ) switches. Switches are the most simple – at the same time, highly non-trivial – examples of constrained networks. They form the core of Internet routers and there is extensive literature dealing with the design and analysis of various switch architectures and scheduling algorithms. A switch is essentially a bipartite network with input ports and output ports. The function of a network switch is to move packets from the input ports to the output ports using the switch fabric, just like the traffic at a traffic junction. Depending on the placement of buffers and the switch fabric, there are mainly two kinds of switch architectures – input queued switches (IQ) and output queued (OQ) switches. As their names suggest, input queued switches have buffers only at input ports, while output queued switches have buffers only at the output ports. The input queued switch has a cross-bar switch fabric that imposes the following natural

constraints: only one packet can be moved from (to) each input (output) port in each time slot. On the other hand, since an output queued switch has buffers only at output ports, packets arriving at the input ports are immediately transferred to their respective output buffers. Thus, there are no scheduling constraints at the switch fabric in an output queued switch. Because of this, the memory in an output queued switch has to operate much faster than the memory in an input queued switch. In most high-speed switches, memory bandwidth is the bottleneck and hence input queued switches are more popular and widely deployed, while output queued switches are idealized versions that are easy to study.

It is clear from the description that scheduling in output queued switches is equivalent to that of single queues. Hence, fair scheduling in output queued switches just corresponds to implementing single queue fair algorithms at different output queues. Unfortunately, such extension is not possible for input queued switches because of the presence of constraints. One approach is to emulate the performance of an OQ switch by means of a IQ switch running with a minimal speedup. An IQ switch is said to be running with a speedup S if it can be scheduled S times in each time slot. This approach was taken by Prabhakar and McKeown (1999) [18] and Chuang, Goel, McKeown and Prabhakar (1999) [19], where they showed that essentially a speedup of 2 is necessary and sufficient for emulating an OQ switch. With the OQ switch operating under any of the various policies like FIFO, WFQ, DWRR, strict priority, etc. fairness can be achieved. Equivalently, if an IQ switch is loaded up to 50% of its capacity and the notion of fairness is defined by policies like FIFO, WFQ, DWRR, strict priority, etc., then by emulating an OQ switch with these policies, it is possible to have fair scheduling for the IQ switch. However, for higher loading this approach will fail due to inability of emulating an OQ switch.

This necessitates the need for defining an appropriate notion of fairness that cleverly, and in a reasonable manner, combines the preferences of packets based on some absolute notions along with the scheduling constraints. In principle, this question is very similar to the question answered by utility maximization based framework for bandwidth allocation in a flow network. In fact, most of the existing literature on fair scheduling algorithms for input-queued switches is concerned with the notion of flow-based fairness. In these approaches, a flow is identified with all the packets corresponding to an input-output pair. There are two main approaches taken in the literature for the design of fair algorithms for IQ switches. One class of fair algorithms implement a fair scheduling scheme at each of the servers in the switch and then carry out an iterative matching. This approach is based on the Distributed Packet Fair Queuing architecture. Examples of this approach include iPDRR proposed by Zhang and Bhuyan (2003) [20], MFIQ proposed by Li, Chen and Ansari (1998) [21], and iFS proposed by Ni and Bhuyan (2002) [22]. This approach completely ignores fairness issues that arise because of scheduling constraints and hence need not guarantee an overall fair bandwidth allocation. In order to overcome this, Hosaagrahara and Sethu (2005) [23] and more recently Pan and Yang (2007) [24] propose algorithms to calculate overall max-min rates of different flows, taking into account contention at all levels. But this approach requires knowledge of rates of flows and hence, the system should either learn these rates or know them a priori.

Thus, most of the literature on the design of fair scheduling algorithms for constrained networks is limited because it either ignores fairness issues caused due to scheduling constraints or directly borrows flow-based fairness notions and allocates bandwidth accordingly. Here, it is important to emphasize the limitations of a flow-based approach: (a) network traffic predominantly contains "short-flows", while flow-based approach requires existence of ever-lasting traffic thereby inducing huge delays when applied naively, (b) flow-based approach requires knowledge of traffic rates, which it may have to learn, (c) our unit of data is a packet and modeling it as a flow is just an approximation, and (d) packets have priorities and they lack explicit utility functions.

In summary, our question is inherently combinatorial which requires dealing with hard combinatorial constraints unlike the resource allocation in a flow network which deals with soft capacity constraints in a continuous optimization setup.

C. Organization

The rest of the paper is organized as follows: Section II describes the model, introduces the notation and states the problem formally. Section III motivates and describes our approach. Section IV takes a digression into Economics literature to explain the ranked election problem. Section V establishes the analogy between the ranked election problem and network scheduling. Sections VI and VII present the main results of this paper. Section VI formally states our algorithm, while Section VII provides the details of the proof of throughput optimality. We provide some simulation results in Section VIII and then finally conclude in Section IX.

II. Model and Notation

We now describe a generic abstract model of a constrained queuing network. The model corresponds to a single-hop network. This generic model describes important special instances like an input queued switch, wireless network limited by interference, congestion control in TCP or even traffic in a road junction. In each of these instances, the model effectively captures the constraints imposed by nature on simultaneous servicing of queues. We will describe the examples of an input queued switch and a wireless network in detail. We focus on these two examples because they encapsulate a large class of scheduling problems.

A. Abstract formulation

Consider a collection of N queues. Time is discrete and is indexed by $\tau \in \{0, 1, \ldots\}$. Each queue has a dedicated exogenous process of packet arrival. The arrival processes of different queues are independent. All packets are assumed to be normalized to unit length. Arrivals to each queue occur according to a Bernoulli process.

The service to the queues is subject to scheduling constraints in that not all queues can be served simultaneously. The

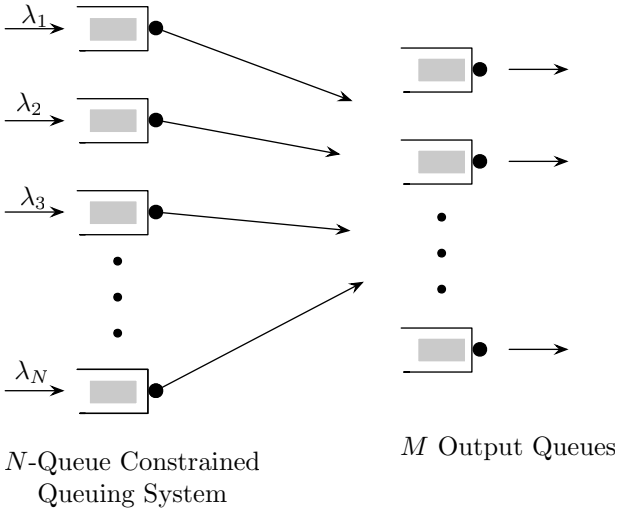

Figure 1. Abstract model of the constrained queuing system

scheduling constraints present are described by a finite set of feasible schedules $\mathscr{S} \subseteq \{0,1\}^N$. In each time slot a feasible schedule $\pi \in \mathscr{S}$ is chosen and queue n is offered a service π_n in that time slot. Since each packet is of unit length, when a non-empty queue receives service, a packet departs from the queue. We assume that $\mathscr{S}$ is monotone i.e., if $\pi \in \mathscr{S}$, then for any $\sigma \leq \pi$ component-wise i.e., $\sigma_n \leq \pi_n$, $\sigma \in \mathscr{S}$. Further, we assume that for each n, there exists a schedule $\pi \in \mathscr{S}$ that serves it i.e., $\pi_n = 1$.

Packets exit the system from any of the M output lines. The lines are assumed to operate at unit speed and hence at most one packet can leave the network from each line in each time slot. Each of the M output lines maintain buffers termed output queues to store packets that are ready to depart. We assume that routing is pre-determined and, hence, the destination output line of each packet is known. After service, each packet is moved to its destination output queue. Each output queue operates using a single queue scheduling policy (eg. First Come First Serve (FCFS), Weighted Fair Queuing (WFQ) etc.). The served packets are queued and served according to the single queue scheduling policy. Fig. 1 illustrates this model.

Under this setup, the problem of scheduling is to choose a *feasible* schedule in each time slot to serve the queues and move the packets to their respective output queues. Since the scheduling policy for each of the output queues can be chosen independently, the problem reduces to that of the constrained collection of queues.

1) Notation: First, some general notation. $\mathbb{R}$ denotes the set of real numbers and $\mathbb{R}_+$ the set of non-negative real numbers i.e., $\mathbb{R}_+ = \{x \in \mathbb{R} \colon x \geq 0\}$. $\mathbb{N}$ denotes the set of natural numbers $\{1, 2, \ldots\}$ and $\mathbb{Z}_+$ the set of non-negative integers $\{0, 1, 2, \ldots\}$. Let $\mathbf{0}$ and $\mathbf{1}$ denote the vectors of 0s and 1s respectively. All the vectors in this paper are length N vectors. Let $\mathbb{1}_{\{\cdot\}}$ denote the indicator function, $\mathbb{1}_{\text{true}} = 1$ and $\mathbb{1}_{\text{false}} = 0$. x^+ denotes $\max\{x, 0\}$ and we use the ℓ_1 norm $|x| = \sum_n x_n$. We also use the standard inner product $\langle a, b\rangle = \sum a_i b_i$.

Recall that we are assuming Bernoulli i.i.d arrivals. Let

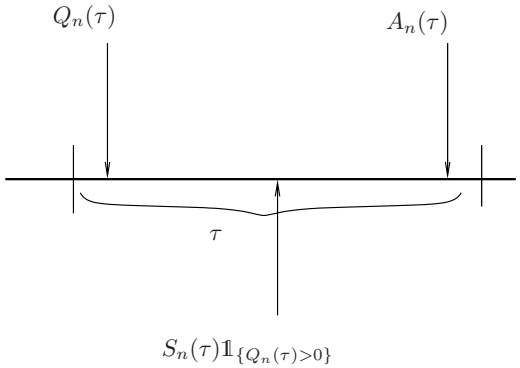

Figure 2. The order in which service and arrivals happen during time slot τ

$A_n(\tau) \in \{0, 1\}$ denotes the number of arrivals to queue n, $n = 1, 2, \ldots, N$, during time slot τ. The arrival rate to queue n is denoted by λ_n i.e., $\Pr(A_n(\tau) = 1) = \lambda_n \ \forall \tau$. $\lambda = (\lambda_n)$ denotes the arrival rate vector. $Q_n(\tau)$ denotes the length of queue n at the beginning of time slot τ. $Q(\tau) = (Q_n(\tau))$ denotes the queue length vector. $S(\tau) \in \mathscr{S}$ denotes the feasible schedule chosen in time slot τ to serve the queues. Without loss of generality, we assume that a feasible schedule is chosen and service happens at the middle of the time slot, and exogenous arrivals occur at the end of the time slot. This is shown in Fig. 2. With $Q_n(\tau)$ denoting the queue length at the beginning of time slot τ, we have:

$$Q_n(\tau + 1) = (Q_n(\tau) - S_n(\tau))^+ + A_n(\tau) \tag{1}$$

Finally, let $D_n(\tau)$ denote the cumulative departure process of queue n i.e.,

$$D_n(\tau) = \sum_{t \leq \tau} S_n(t)\mathbb{1}_{\{Q_n(t)>0\}}. \tag{2}$$

2) Definitions: We now introduce some definitions. We call a system rate stable, or simply stable in this paper, if the following holds with probability 1: for $1 \leq n \leq N$,

$$\lim_{\tau \to \infty} \frac{D_n(\tau)}{\tau} = \lambda_n \tag{3}$$

An arrival rate vector $\lambda = (\lambda_n)$ is called admissible if $\exists$ a scheduling policy under which the queuing network loaded with λ has a queue size process $Q_n(\tau)$ such that

$$\limsup_{\tau} \mathbb{E}\left[|Q(\tau)|\right] < \infty \tag{4}$$

Let Λ denote the set $\left\{\lambda \in \mathbb{R}_+^N \colon \lambda \text{ is admissible}\right\}$. Λ is called the throughput region or capacity region of the network. Tassiulas and Ephremides (1992) [25] proved that:

$$\operatorname{relint} \operatorname{co}(\mathscr{S}) \subseteq \overline{\operatorname{co}(\mathscr{S})} \tag{5}$$

where $\operatorname{co}(\mathscr{S})$ denotes the convex hull of $\mathscr{S}$ i.e., $\left\{\mu \in \mathbb{R}_+^N \colon \mu = \sum_i \alpha_i \pi^i, \alpha_i \geq 0, \pi^i \in \mathscr{S}, \sum_i \alpha_i \leq 1\right\}$. $\operatorname{relint} \operatorname{co}(\mathscr{S})$ denotes the relative interior of $\operatorname{co}(\mathscr{S})$ and $\overline{\operatorname{co}(\mathscr{S})}$ denotes the closure of $\operatorname{co}(\mathscr{S})$. Denote $\operatorname{relint} \operatorname{co}(\mathscr{S})$ by Λ'. It was also shown by Tassiulas and Ephremides (1992) [25] that:

$$\Lambda' = \Big\{\mu \in \mathbb{R}_+^N : \mu = \sum_i \alpha_i \pi^i; \quad \alpha_i \geq 0, \pi^i \in \mathscr{S}, \sum_i \alpha_i < 1\Big\}.$$

We call a scheduling algorithm throughput optimal if $\forall\ \lambda \in \Lambda'$, the system is rate-stable.

3) Constraint free network (CFN): We now introduce the notion of a Constraint Free Network (CFN). A constraint free network is defined as a queuing network in which all the queues can be served simultaneously. Therefore, for a CFN, $\mathscr{S} = \{0,1\}^N$. Thus, scheduling just entails moving arriving packets immediately to their respective destination output queues.

As discussed in Related Work (Section I-B), fairness is well understood for single queues. Therefore, using a single queue fair scheduling scheme for each of the output queues yields a fair scheduling algorithm for the CFN. We assume throughout that a CFN is operating using a fair scheduling algorithm. Along with a CFN we define a shadow CFN as follows: Given a constrained queuing network $\mathcal{N}$, a CFN $\mathcal{N}'$ with the same number of queues and fed with copies of exogenous arrivals to $\mathcal{N}$, is called the shadow CFN of $\mathcal{N}$.

We conclude this section with a brief motivation for the definition of CFN. This also serves as a preview to our approach to the problem. As mentioned earlier, the difficulty in designing a fair scheduling algorithm for networks arises because of the presence of constraints. In the absence of such constraints, the notion of fairness is equivalent to that of a single queue. Thus, we define an ideal network that is constraint-free, whose performance we want to emulate. This is in some sense the best we can do in providing fairness and, thus, serves as a benchmark for defining notions of fairness.

B. Scheduling algorithms

We consider the problem of designing scheduling algorithms for constrained queuing networks. A scheduling scheme or algorithm is a procedure whereby an appropriate feasible schedule is chosen in each time slot. In this paper, we will be interested in a class of scheduling algorithms termed the Maximum Weight (MW) Scheduling Algorithms. In general, a maximum weight algorithm works as follows: In each time slot τ, each queue n is assigned a weight $\omega_n(\tau)$. This weight is usually – but not necessarily – a function of the queue size $Q_n(\tau)$. Then, the algorithm chooses the schedule with the maximum weight i.e.,

$$S(\tau) = \arg\max_{\pi \in \mathscr{S}} \langle \omega(\tau), \pi \rangle \tag{6}$$

A feasible schedule $\pi \in \mathscr{S}$ is said to be maximal if $\forall \sigma \in \mathscr{S}$, $\pi \not\leq \sigma$ component wise. The set of all maximal feasible schedules will be denoted by $\mathscr{S}_{\max}$. It is reasonable to assume that we want to serve as many queues as possible in each time slot. Therefore, when the algorithm chooses a feasible schedule π, we serve the queues according to a maximal schedule $\mu \in \mathscr{S}_{\max}$ such that $\mu \geq \pi$.

Remark. An important special instance of the MW scheduling algorithm is the one with queue sizes as the weights i.e., $\omega_n(\tau) = Q_n(\tau)$. In their seminal work, Tassiulas and Ephremides (1992) [25] (and independently McKeown et. al. (1996) [26]) showed that the MW algorithm with queue sizes as weights is rate stable.

Since these results, there has been a significant work on designing high-performance, implementable packet scheduling algorithms that are derivatives of maximum weight scheduling, where weight is some function of queue-sizes. All of these algorithms are designed to optimize network utilization as well as minimize delay (for example, see recent work by Shah and Wischik (2006) [27]). However, these algorithms ignore the requirement of fairness. Specifically, it has been observed that the maximum weight based algorithm can lead to unwanted starvation or very unfair rate allocation when switch is overloaded (for example, see work by Kumar, Pan and Shah (2004) [28]). We provide a simple example of a switch (detailed description given in the next subsection) to illustrate this: Consider a 2 × 2 switch with arrival rate matrix $\lambda_{11} = \lambda_{21} = 0$, $\lambda_{12} = 0.6$, $\lambda_{22} = 0.5$. Here λ_{ij} corresponds to the arrival rate of traffic at input port i for output port j. Under this loading, output port 2 is overloaded. If OQ switch has Round Robin (or Fair) policy at output 2 so that traffic from both inputs is served equally, then input 1 will get rate 0.5 and input 2 will get rate 0.5 from output 2. However, the maximum weight matching policy, with weight being queue-size (or for that matter any increasing continuous function of queue-size), the algorithm will try to equalize lengths of queues at both inputs. Therefore, input 1 will get service rate 0.55 while input 2 will get service rate 0.45 from output 2.

C. Input queued switch

We now discuss an input queued switch as a special instance of the abstract model that we have described. As mentioned before, a switch is present at the core of an Internet router. A router moves packets from input ports to output ports. Based on the final destination of the arriving packet, a router determines the appropriate output port and then transfers the packet accordingly. The transfer of packets to the corresponding output ports is called switching.

There are various switching architectures, but we discuss the one that is commercially the most popular. Consider an input queued switch containing M input ports and M output ports. The queues at the output ports correspond to the output queues mentioned above, and hence we retain the notation M for the number of output of queues; since we are considering only switches with equal number of input and output ports, the number of input ports is also M . Packets arriving for input port i and destined for output port j are stored at input port i in Q_{ij}. Note that for a switch, it is convenient to denote the queues as Q_{ij} instead of as Q_n, as we do in the generic model. Further, note that the total number of queues $N = M^2$. The switch transfers packets from input ports to output ports using the switching fabric. The crossbar switching fabric implemented in an input queued switch imposes the following constraints on packet transfer from input to output ports: in

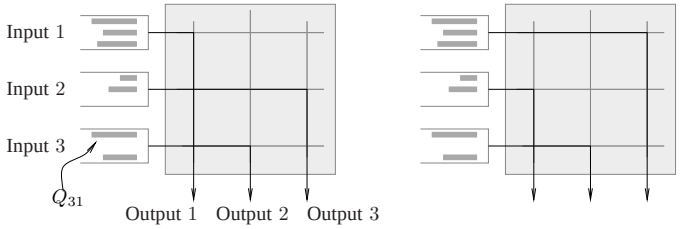

Figure 3. A 3 port input queued switch showing two different possible matchings.

each time slot, each input port can transmit at most one packet and each output port can receive at most one packet. Therefore, feasible schedules are matchings from input to output ports. This is illustrated in Fig. 3. The left and right hand figures illustrate two different possible matchings.

The scheduling algorithm in the input queued switch chooses an appropriate matching in each time slot. To link back to the abstract model that we described, note that an M port switch has $N = M^2$ constrained queues; for the queues we use the notation $\cdot_{ij}$ and not $\cdot_n$ for all terms to clearly reference the input and output ports. The set of all feasible schedules $\mathscr{S}$ corresponds to the set of all matchings in an $M \times M$ bipartite graph:

$$\mathscr{S} = \Big\{\pi = (\pi)_{ij} \in \{0,1\}^{M\times M} : \sum_{k=1}^{M} \pi_{ik} \leq 1; \sum_{k=1}^{M} \pi_{kj} \leq 1, 1 \leq i,j \leq M\Big\}.$$

Packets leave the switch from their respective output ports and hence the output ports correspond to the output queues. Since at most one packet arrives at each output port in each time slot, the packet immediately departs from the output queue. Thus, scheduling reduces to choosing an appropriate matching in each time slot. We point an interested reader to [19] for a more detailed exposition on switch architectures.

D. Wireless networks

We now consider wireless networks as a special instance of the abstract model. Consider a collection of devices (eg. sensor nodes, WiFi nodes, etc.) that are using the wireless medium to transmit messages. The devices share the same frequency to transmit and hence interfere when they transmit simultaneously. Because of power constraints, only devices that are close to each other geographically can communicate. This is often modeled by a graph with a node for each device and an edge between two nodes if they can communicate.

Power constraints also limit interference to nodes that are close to each other; in other words, only nodes that are connected by an edge interfere. Therefore, the graph also models the interference constraints on scheduling. In other words, transmission to a node is successful only if none of its neighbors in the graph is transmitting at the same time. This model of interference is termed the *independent set* model of interference.

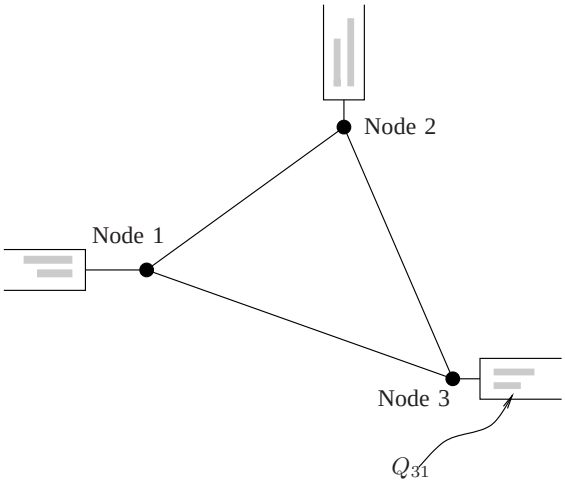

Figure 4. A 3 node wireless network

We assume that the network is modeled as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V}$ denoting the node set $\{1, 2, \ldots, N\}$ and $\mathcal{E}$ denoting the directed edge set $\{(i,j)\colon i \text{ communicates with } j\}$. Each node i maintains a queue Q_{ij} for each of its neighbors j. We assume a single hop network in which packets arrive at nodes, get transmitted to one of the neighbors and then leave the network through output queues. Fig. 4 illustrates a wireless network with three nodes operating under interference constraints.

In this setup, the scheduling problem is to decide which directed links will be active simultaneously. Constraints limit feasible schedules to those in which none of the neighbors of a receiver is transmitting; in other words, if link (i,j) is active then none of the links in the set $\{(l,k) \in \mathcal{E}\colon (l,j) \in \mathcal{E} \text{ OR } (j,l) \in \mathcal{E}\}$ should be active. For each network represented by graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we can construct a conflict graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ with a node for each of the directed links and an edge between two links if they cannot be active simultaneously. The feasible schedules then reduce to independent sets in the conflict graph. Formally,

$$\mathscr{S} = \Big\{\pi \in \{0,1\}^{|\mathcal{V}'|} : \pi_i + \pi_j \leq 1, \text{ for all } (i,j) \in \mathcal{E}'\Big\} \tag{7}$$

It should be noted that using the conflict graph, more general constraints in the network can be modeled as independent set constraints. Thus, the model we are considering encapsulates the essence of a large class of scheduling problems.

III. Our Approach

Network resources are shared by different users and our goal is to design a scheduling algorithm that allocates resources in a fair manner. Before we can design such an algorithm, there is a need to give a precise definition of the users and the resources of the network. Traditionally, different traffic flows were considered the users and the bandwidth allocated to them the resource of the network. Link capacity constraints limited the total amount of resources available, and each flow was allocated its "fair" share of bandwidth. This is the basis of the *utility maximization* framework in which the utility of

each flow was a function of the allocated bandwidth – the more the bandwidth allocated, the greater the utility. Different utility functions yield different fairness criteria. An inherent limitation of this approach is that it considers entire flows as users, disregarding the fact that flows are not continuous but are composed of packets. Moreover, bandwidth is a resource that is allocated on an average over a long period of time assuming that flow statistics remain constant over such long periods.

We overcome this limitation by treating the Head-of-Line (HoL) packet of each flow as the user of the network resources (We assume that each queue is associated with a flow and hence we use these terms interchangeably). This takes into account the packetized nature of a flow and is a consequence of the realization that in each time slot the decision is whether to serve the HoL packet or not, unlike the case of a continuous flow that can be given fractional service. Therefore, utilities should correspond to HoL packets and not entire flows. With HoL packets as the users, the network resource becomes the service they receive in each time slot. The network resource is limited by the constraint set $\mathscr{S}$ and the algorithm should choose a feasible schedule $\pi \in \mathscr{S}$ in a manner that is "fair" to all the HoL packets. Inspired by the utility maximization framework, we could define utility functions for the HoL packets and choose a feasible schedule that maximizes the overall utility. But, there is no natural choice of the utility function and hence we take a different approach.

We begin with the realization that packets do not have natural utility functions, but they do have a natural preference order of the feasible schedules. For each packet, there are two classes of schedules – one class containing all schedules that serve it and the other containing all schedules that do not. The packet is indifferent to all the schedules in the same class and the preference relation between schedules in different classes depends on how "urgently" the packet wants to get served. Fair scheduling now reduces to combining individual preferences in a fair manner to come up with a "socially" preferred schedule. This is equivalent to a ranked election problem: HoL packets (queues) are voters, schedules are candidates and each packet has a preference list of the schedules (refer to Section IV for more details on the ranked election problem). The problem of ranked election is very well studied in the Economics literature (also called the theory of social choice). In their seminal work in 1952, Goodman and Markowitz [5] prove that under certain socially desirable postulates (detailed in Section IV), when the voters have cardinal (quantitative) preferences over candidates, a simple function of those quantitative preferences yields a uniquely preferred outcome.

In order to use the Goodman and Markowitz result, we require not just relative preferences, but quantitative preferences over the feasible schedules. By quantitative preferences we mean that each packet assigns numerical weights to schedules; the higher the weight, the more preferred the schedule. In principle, the packet can assign any weights consistent with its relative order of preferences to obtain quantitative preferences over the schedules. But it is important to realize that the choice of quantitative preference is crucial for obtaining the practically desired benefits of the fair scheduling. In our setup, each packet has two classes of schedules: one that it prefers to the other while being indifferent to schedules within the same class. Therefore, the packet assigns the same weight to all the schedules within the same class. Since only relative preferences matter, we assume that each packets assigns the same weight 0 to all schedules in the class it does not prefer. Assigning quantitative preferences now reduces to choosing a weight for each packet to assign to the class of schedules it prefers.

One feasible option would be to use queue sizes as weights. The problem with this choice is that it is oblivious to flow identities and is susceptible to manipulation (a flow can gain advantage by overloading system with packets resulting in large queue sizes). Another option would be to use the age (waiting time in the system) of the packet. This choice is still oblivious to flow and packet identities and it is difficult to provide QoS by giving priority to one flow (packet) over other . We overcome these problems by using the idea of emulation of the shadow CFN. As mentioned in Related Work (Section I-B), one way of designing a fair scheduling algorithm would be to perfectly emulate a CFN using a fair queuing policy at each of the output queues. But this results in a loss of throughput of the system. Therefore, our approach would be to emulate the shadow CFN as closely as possible. In this spirit, we use a function of the departure time of the packet from the shadow CFN as the weight; the earlier the departure time, the higher the weight. The details of the exact function used are covered in Sections V and VI.

We now tie this back to the utility maximization framework. Using the Goodman and Markowitz algorithm with the above choice of weights yields a MW style algorithm with the weight of each queue equal to the weight assigned to the packet. This is identical to the result we obtain by using the assigned weights as utilities of packets and choosing a schedule that maximizes overall utility. Therefore, our algorithm yields utility functions for packets that can be used in the utility maximization framework. This rather surprising result connects our approach back to utility maximization very nicely.

We then establish that such an algorithm is throughput optimal under the standard stochastic model of a network. To prove throughput optimality (rate stability to be precise), we use an appropriate quadratic Lyapunov function. However, we cannot use the standard stability proof technique based on Foster's criterion because the Lyapunov function is not a function of queue-sizes, but is function of *preferences* derived from the shadow CFN. This makes the analysis rather non-trivial.

To explain the consequences of our algorithm on fair emulation, we present simulations for algorithms based on FIFO OQ switch. Intuitively, our fair algorithm should be able to reduce the queue-size (or delay) as well as get rid of starvation caused by well-known throughput optimal algorithms. Our simulation results clearly confirm this intuition.

IV. Ranked Election

In this section we take a digression into Economics literature to describe the ranked election problem.

Definition 1 (Ranked election)**.** There are M voters that vote for C candidates. Vote of each voter consists of a ranking (or permutation) of all C candidates. These votes can additionally carry quantitative values associated with their preferences. Let a_{mc} denote the value voter m gives to candidate c, for $1 \leq m \leq M$, $1 \leq c \leq C$. The goal of the election is to relative order all the C candidates as well as produce the ultimate winner in a manner that is consistent with the votes.

The key for a good election lies in defining consistency of the outcome of election with votes. The following are canonical postulates that are used in the literature on ranked election:

P1. Between any two candidates c and c', suppose that none of the M voters prefers c' to c and at least one voter prefers c to c'. Then c' should not be ranked higher than c in the output of the election. This property corresponds to the economic notion of weak Pareto optimality.
P2. Suppose the voters are renumbered (or renamed) while keeping their votes the same. Then the outcome of election should remain the same. In other words, the election outcome is blind to the identity of the voters, that is election outcome is symmetric.
P3. Now, consider the setup when the votes are cardinal (i.e., quantitative). Suppose candidate c is preferred to c' by the election. Then, by adding the same fixed constant to all a_{mc} and $a_{mc'}$ for $1 \leq m \leq M$, the relative order of candidates c and c' should not change. This makes sense because what matters is the difference in preference levels for the two candidates, not the actual values.

In the absence of cardinal (or quantitative) preferences, the question of ranked election with postulates P1, P2 (and some additional postulates) was first studied by Arrow (1951) [29]. In his celebrated work, he established the (then) very surprising impossibility of the existence of any election scheme that satisfies P1, P2 (and additional postulates) simultaneously. We note that this result has been an important corner stone in the field of theory of social choice.

Subsequent to Arrow's impossibility result, many economists started looking for positive results. Among many other celebrated results, the result that is relevant to this paper is that of Goodman and Markowitz (1952) [5]. They showed that if voters have cardinal preferences, as in our setup, then there is a unique ordering of candidates that satisfies P1-P2-P3 simultaneously. To describe their result, consider the following: let the net score of a candidate$c$ be $s_c = \sum_{m=1}^{M} a_{mc}$. Goodman and Markowitz obtained the following remarkable result.

Theorem 2. *Suppose the scores of all candidates are distinct. Rank candidates as follows: candidate c has higher ranking than c' if and only if $s_c > s'_c$. This ranking satisfies postulates P1-P2-P3. Further, this is the only such ranking.*

For a proof of this result, we refer the reader to [5].

V. Analogy between fair scheduling and ranked election

In this section, we motivate our fair scheduling algorithm by establishing an equivalence between fair scheduling in a constrained queuing network $\mathcal{N}$ and the problem of ranked election. In our context, the packets (queues) are the voters and the feasible schedules $\pi \in \mathscr{S}$ are the candidates. In order to use the Goodman and Markowitz setup, we need to derive preferences for packets over schedules. For each packet, there are two classes of schedules – one class containing all schedules that serve it and the other containing all schedules that do not. The packet is indifferent to all the schedules in the same class. Since only the relative weights matter, we assume that a packet assigns a weight of 0 to all schedules that do not serve it.

We derive preferences for packets over schedules that serve them from the corresponding shadow CFN $\mathcal{N}'$ that is operating with a single queue fair scheduling policy at each of its output queues. As defined before, a copy of every packet arriving to the network $\mathcal{N}$ is fed to the shadow CFN $\mathcal{N}'$. That is, (a copy of) a packet arriving at queue n for output queue m of $\mathcal{N}$ immediately joins the output queue m in $\mathcal{N}'$. The departures from the output queues of $\mathcal{N}'$ happen according to an appropriate fair scheduling policy, say $\mathcal{P}$, such as strict priority scheme, last-in-first-out or simply first-in-first-out. Specifically, our objective in assigning preferences is to have the departures of packets from $\mathcal{N}$ be as close as possible to the departures from the corresponding shadow CFN $\mathcal{N}'$. Ideally, we want $\mathcal{N}$ to exactly emulate $\mathcal{N}'$ i.e., we want the departure times of packets from both the networks to be exactly the same. However, we settle with approximate emulation because, as shown by Chuang et. al. [19], exact emulation is not possible at speedup 1. Since the preferences of packets are chosen from $\mathcal{N}'$ and these preferences are combined in a fair manner, the fair scheduling polices at the output queues of $\mathcal{N}'$ can now be chosen according to the desired requirements.

Based on the above discussion, our approach is to use a value that is a function of the departure time of the packet from$\mathcal{N}'$ – the earlier the departure time, the higher the value assigned. More specifically, let p_n^τ denote the HoL packet in queue n of network $\mathcal{N}$ at time τ. Let $d_n(\tau)$ denote the departure time of p_n^τ from $\mathcal{N}'$. For the following discussion we assume that the queue is non-empty and hence $d_n(\tau)$ is well defined. We defer the discussion of empty queues to the next section. Now, each queue n assigns a value of $\tau - d_n(\tau)$ to all the schedules that serve it. (The choice of $\tau - d_n(\tau)$ seems arbitrary, when we could have taken any decreasing function of $d_n(\tau)$. Indeed we can, though it should have some "nice" properties to maximize throughput. Details are in Section VII). This completes the equivalence.

Taking a closer look at the weight $\tau - d_n(\tau)$, note the following. Suppose at time τ the packet p_n^τ is already late i.e, $d_n(\tau) < \tau$. In this case, the weight $\tau - d_n(\tau) > 0$, which means that p_n^τ prefers all schedules that serve it to all the schedules that do not by weight $\tau - d_n(\tau)$. Thus, the more delayed the packet is, the higher the weight it assigns. On

the other hand, when the packet is not late i.e., $d_n^\tau > \tau$, p_n^τ prefers schedules that do not serve it in order to give a chance to packets that are late to get served.

Now, with the above assignment of values to each schedule by each queue, the value of a schedule $\pi \in \mathscr{S}$ is given as:

$$\begin{aligned}\mathsf{value}(\pi) &= \sum_{n=1}^{N} \pi_n(\tau - d_n(\tau)) \\ &= \langle \tau - d(\tau), \pi \rangle\end{aligned}$$

The postulates P1-P2-P3 translate into the following postulates for network scheduling.

P1′. Between any two schedules n_1 and n_2, suppose that none of the N HoL packets prefer n_2 to n_1 and at least one HoL packet prefers n_1 to n_2. Then, we should not choose n_2.

P2′. For given HoL packets, let π be the outcome of the election as per the above preferences for schedules. Then, by renumbering queues while retaining the same HoL preferences, the outcome of election should be only renumbered π. In other words, the election does not give unfair priority to any port and thus is symmetric in its inputs.

P3′. Suppose schedule π^1 is preferred to π^2 by the election. By adding the same fixed constant to $\tau - d_n(\tau)$ for all n, the outcome of the election should remain unchanged.

The election algorithm of Goodman and Markowitz suggests that the following schedule $S(\tau)$ should be chosen:

$$S(\tau) \in \arg \max_{\pi \in \mathscr{S}_{\max}} \langle \tau - d(\tau), \pi \rangle$$

VI. Most Urgent Cell First (MUCF(f)) Algorithm

Based on the discussion in the previous section, we propose a fair scheduling algorithm called the *most urgent cell first* algorithm. According to this algorithm, packets are scheduled according to the maximum weight schedule (MWS) with urgencies of queues as weights with the urgency $U_n(\tau)$ of queue n defined as $\tau - d_n(\tau)$ if it is non-empty i.e., the more "late" the packet is, the higher is its urgency. If queue n is empty we define its urgency as $-\max\{0, -\min_{n:\, Q_n(\tau)>0} U_n(\tau)\}$. Note that according to this definition, the weight assigned to a schedule by an empty queue is always less than or equal to the weight assigned by any non-empty queue in the system. Therefore, as desired, the fair schedule chosen by the algorithm tries to minimize the number of empty queues it serves. Of course, as mentioned in the previous section, we could have used any increasing function of the urgency. In particular, suppose $f\colon \mathbb{R} \to \mathbb{R}$ denotes a non-decreasing bi-Lipschitz continuous function with Lipschitz constant $\rho > 1$ i.e., for any $x, y \in \mathbb{R}$, $1/\rho\,|x-y| \le |f(x) - f(y)| \le \rho\,|x-y|$. Without loss of generality, we assume that $f(0) = 0$; thus, $f(x) \ge 0$ for $x \ge 0$. The MUCF(f) algorithm now chooses the MWS with weight of queue n equal to $U_n^f(\tau)$, defined as $f(U_n(\tau))$. Formally, the three components of the algorithm are as follows:

1) The arriving packets are queued according to the **FIFO** queuing policy in each of the N constrained queues.
2) For a packet p in $\mathcal{N}$, let $d(p)$ denote its departure time from $\mathcal{N}'$. Then, the arriving packets in each of the M output queues are queued in the order of their respective departure times from $\mathcal{N}'$. More formally, in every output queue m, a packet p will be ahead of every packet p' that satisfies $d(p') > d(p)$.
3) In each time slot τ , the algorithm chooses a feasible schedule $S(\tau)$ from $\mathscr{S}$ using a MW criterion as follows:

$$S(\tau) \in \arg \max_{\pi \in \mathscr{S}_{\max}} \left\langle U^f(\tau), \pi \right\rangle \tag{8}$$

VII. Throughput of MUCF(f) Algortihm

The previous section described how we arrived at MUCF algorithm as a fair algorithm based on preferences obtained from a shadow CFN. As established in the previous section, Theorem 2 implies that MUCF is the only algorithm that satisfies the desirable postulates P1′-P2′-P3′. In this section, we state and prove the throughput optimality property of the MUCF algorithm. The proof of the algorithm is non-traditional and requires new techniques that may be of interest for analysis of such non-queue based weighted algorithms.

Theorem 3. *Consider a constrained queuing system with an arbitrary set of constraints $\mathscr{S}$. Suppose the system is loaded with an i.i.d. Bernoulli arrival process, and is operating under the MUCF(f) algorithm with$f(\cdot)$ a bi-Lipschitz function. Then, if the rate vector is strictly admissible, the queuing network is rate stable.*

Before we prove Theorem 3 we need the following notation and lemmas.

Notation. First, some useful notation. Consider the HoL packet p_n^τ of queue n in network $\mathcal{N}$ at the beginning of the time slot τ . As before, let $a_n(\tau)$ be its time of arrival and $d_n(\tau)$ be the time of its departure from $\mathcal{N}'$, $U_n(\tau)$ be its urgency as defined above, and $W_n(\tau)$ be its waiting time (i.e., $\tau - a_n(\tau)$ if the queue is non-empty and 0 if it is empty). Let $W_n^f(\tau)$ denote $f(W_n(\tau))$ and $F(y)$ denote $\int_o^y f(x)dx$. Also, define $\Delta_n(\tau)$ as $W_n(\tau) - U_n(\tau)$. We note that if queue n is empty, then $W_n(\tau) = 0$ and $U_n(\tau)$ is as defined above. Hence, $\Delta_n(\tau)$ is always non-negative. Let B_m^k, $m = 1, 2, \ldots, M$, denote the length of the k^{th} busy cycle at output queue m in $\mathcal{N}'$. Finally, for any function $g\colon \mathbb{R} \to \mathbb{R}$ and a vector $v \in \mathbb{R}^N$, $g(v) \in \mathbb{R}^N$ denotes $(g(v_n))$. Before we state the lemmas and prove Theorem 3, note the following property of $F(\cdot)$:

$$F(0) = 0, \quad F(y) \ge 0, \text{ for all } y \in \mathbb{R}. \tag{9}$$

The equality $F(0) = 0$ follows directly from the definition of $F(\cdot)$. Coming to $F(y) \ge 0$, note that since $f(\cdot)$ is non-decreasing and $f(0) = 0$, it follows that $f(x) \le 0$ for $x \le 0$ and $f(x) \ge 0$ for $x > 0$. Hence, for $y > 0$, $F(y) = \int_0^y f(x)dx \ge 0$ since $f(x) \ge 0$ for $x > 0$. Similarly, for $y < 0$, $F(y) = \int_0^y f(x)dx = \int_y^0 (-f(x))dx \ge 0$ since $-f(x) \ge 0$ for $x < 0$.

Lemma 4. *Let $L(\tau) := \langle F(W(\tau)), \lambda\rangle = \sum_n F(W_n(\tau))\lambda_n$. Then, under the MUCF(f) algorithm with $f(\cdot)$ a bi-Lipschitz function and λ being strictly admissible, there exists an $\varepsilon > 0$ such that*

$$\mathbb{E}\left[L(\tau+1) - L(\tau)\right] \leq -\varepsilon\mathbb{E}\left[|W(\tau)|\right] + 2\mathbb{E}\left[|\Delta(\tau)|\right] + K,$$

for some constant K.

Lemma 5. *Under the MUCF(f) algorithm, with $f(\cdot)$ a bi-Lipschitz function and λ being strictly admissible, suppose $Z_\tau = \frac{1}{\tau\log\tau}\sum_{i=1}^{\tau}|W(i)|$ and $\mathbb{E}[Z_\tau] \leq O(1) < \infty$ for all τ. Then, we must have*

$$\Pr\left(\lim_{\tau\to\infty}\frac{1}{\tau}|W(\tau)| = 0\right) = 1. \tag{10}$$

Lemma 6. *Let $\Theta_m(\tau)$ denote $\max_{0\leq k\leq\tau} B_m^k$, for $m = 1, 2, \ldots, M$. Then, under a strictly admissible λ with the output queues of $\mathcal{N}'$ operating under a Work Conserving Policy (WCP), the following is true for all tand $1 \leq m \leq M$,*

$$\mathbb{E}\left[\Theta_m(\tau)\right] \leq O(\log\tau)$$

We will first prove the result of Theorem 3 assuming the results of Lemmas 4 and 6, and defer their proof until after the proof of Theorem 3.

Proof of Theorem 3: We first note that if queue n is non-empty then

$$\Delta_n(\tau) \leq \max_{0\leq k\leq\tau} B_m^k \tag{11}$$

where m is the destination output queue of packet p_n^τ. This is true because when queue n is non-empty, $\Delta_n(\tau)$ denotes the waiting time of p_n^τ in its destination output queue in $\mathcal{N}'$, and hence cannot be more than the length of the busy cycle it belongs to. Since there can be at most τ busy cycles up to time τ and p_n^τ arrived to $\mathcal{N}'$ before τ, (11) should be true. Therefore, from lemma 6 and (11) it follows that:

$$\mathbb{E}\left[\Delta_n(\tau)\right] \leq O(\log\tau) \tag{12}$$

If queue n is empty, then by definition it follows that either $\Delta_n(\tau) = 0$ or $\Delta_n(\tau) = d_{n'}(\tau) - \tau$, for some queue n' that is non-empty. Since, $\Delta_l(\tau) \geq 0\ \forall l, \tau$, we have from (12) that $\mathbb{E}\left[\Delta_n(\tau)\right] \leq \mathbb{E}\left[\Delta_{n'}(\tau)\right] \leq O(\log\tau)$. Hence, (12) is valid even for empty queues and it thus follows that:

$$\mathbb{E}\left[|\Delta_n(\tau)|\right] \leq O(\log\tau) \tag{13}$$

From lemma 4 and (13), we obtain the following:

$$\mathbb{E}\left[L(\tau+1) - L(\tau)\right] \leq -\varepsilon\mathbb{E}\left[|W(\tau)|\right] + O(\log\tau) + K, \tag{14}$$

Telescopic summation of (14) from $1, 2, \ldots, \tau$, we obtain (after cancellations),

$$\mathbb{E}\left[L(\tau+1)\right] \leq \mathbb{E}\left[L(0)\right] - \varepsilon\mathbb{E}\left[\sum_{i=1}^{\tau}|W(i)|\right] + O(\tau\log\tau) + \tau K, \tag{15}$$

Now, the network starts empty at time 0. Therefore, $\mathbb{E}\left[L(0)\right] = 0$. Further, $L(\cdot)$ is non-negative function. Therefore, (15) gives us

$$\varepsilon\mathbb{E}\left[\sum_{i=1}^{\tau}|W(i)|\right] \leq O(\tau\log\tau) + \tau K. \tag{16}$$

Dividing both sides by $\varepsilon\tau\log\tau$, we obtain

$$\mathbb{E}\left[\frac{1}{\tau\log\tau}\sum_{i=1}^{\tau}|W(i)|\right] \leq O(1). \tag{17}$$

Let $X_\tau = \frac{1}{\tau}\sum_{i=1}^{\tau}|W(i)|$ and $Z_\tau = \frac{X_\tau}{\log\tau}$. From (17), we have $\mathbb{E}\left[Z_\tau\right] \leq O(1) < \infty$ for all τ. It now follows from Lemma 5 that

$$\Pr\left(\lim_{\tau}\frac{1}{\tau}|W(\tau)| = 0\right) = 1 \tag{18}$$

Using (18), we complete the proof of rate stability of the algorithm as follows. At time τ, the waiting time of the HoL packet of queue n is $W_n(\tau)$. Because of FIFO policy and at most one arrival per time slot, we have that the queue-size of queue n at time τ, $Q_n(\tau) \leq W_n(\tau)$. From (18), we have that

$$\lim_{\tau\to\infty}\frac{Q_n(\tau)}{\tau} = 0, \text{ with probability 1.} \tag{19}$$

Now, $Q_n(\tau)$ observes the following dynamics:

$$Q_n(\tau) = Q_n(0) + \sum_{t\leq\tau}A_n(t) - D_n(\tau), \tag{20}$$

where the second term on RHS is the cumulative arrival to queue n till time τ while the third term is the cumulative departure from queue n till time τ. By strong law of large numbers (SLLN) for Bernoulli i.i.d. process we have that:

$$\lim_{\tau\to\infty}\frac{1}{\tau}\sum_{t\leq\tau}A_n(t) = \lambda_n.$$

Using this along with (19) and (20), we obtain

$$\lim_{\tau\to\infty}\frac{D_n(\tau)}{\tau} = \lambda_n, \quad \text{with probability 1, } \forall n.$$

This completes the proof of Theorem 3. ∎

Proof of Lemma 5: Suppose (10) is not true. Then, since $|W(\tau)| \geq 0$ we have that for some $\delta > 0$,

$$\Pr\left(|W(\tau)| \geq \delta\tau, \text{ i.o.}\right) \geq \delta, \tag{21}$$

where "i.o." means infinitely often. Now if $|W(\tau)| \geq \delta\tau$, then there exists an HoL packet that has been waiting in the network for time at least $\delta\tau/N$. This is true because $|W(\tau)|$ is the sum of waiting times of at most N HoL packets. Call this packet p. This packet must have arrived at time $\leq \tau - \delta\tau/N = \tau(1-\delta)N^{-1}$. Since waiting time of a packet increases only by 1 each time-slot, the waiting time of packet p must be at least $0.5\delta\tau/N$ in time interval $[\tau_1, \tau]$, where $\tau_1 = \tau - 0.5\delta\tau N^{-1} = \tau(1 - 0.5\delta N^{-1})$. Now, consider any time $\tau' \in [\tau_1, \tau]$. The packet waiting at the HoL of the queue that contains p must have waiting time higher than that of p due to the FIFO ordering policy. Therefore, the contribution to $|W(\tau')|$ by HoL packets of the queue that contains packet p is at least $0.5\delta\tau N^{-1}$. Therefore, we obtain

$$\sum_{\tau'=\tau_1}^{\tau}|W(\tau')| \geq (\tau - \tau_1)\frac{\delta\tau}{2N} = \frac{\delta^2\tau^2}{4N^2} \tag{22}$$

Therefore, by the definition of X_τ and non-negativity of $|W(\cdot)|$, we have the following logical implication:

$$|W(\tau)| \geq \delta\tau \implies X_\tau \geq \frac{\delta^2\tau}{4N} \tag{23}$$

Thus, if (21) holds then by (23) we have

$$\Pr\left(X_\tau \geq \frac{\delta^2\tau}{4N^2}, \text{ i.o.}\right) \geq \delta. \tag{24}$$

Now observe the following relation of X_t: since $|W(\cdot)| \geq 0$,

$$X_{t+1} \geq \left(1 - \frac{1}{t+1}\right) X_t.$$

For any integer $L > 0$ and any integer $t' \in [t, t+L]$, we can now write

$$\begin{aligned} X_{t+L} \geq \prod_{i=t'+1}^{t+L} \left(1 - \frac{1}{i}\right) X_{t'} &\geq \prod_{i=t+1}^{t+L} \left(1 - \frac{1}{i}\right) X_{t'} \\ &\geq \left(1 - \frac{1}{t}\right) X_{t'}. \end{aligned}$$

Now, for any $\alpha > 1$, let $L = \lfloor t\alpha \rfloor - t$, which implies that $L \leq (\alpha - 1)t$. We can then write for any integer $t' \in [t, \alpha t]$,

$$X_{\lfloor t\alpha \rfloor} \geq \left(1 - \frac{1}{t}\right)^L X_{t'} \geq \left(1 - \frac{1}{t}\right)^{(\alpha-1)t} X_{t'}.$$

Since

$$\left(1 - \frac{1}{t}\right)^{(\alpha-1)t} \approx \exp(-\alpha + 1),$$

taking $\alpha = 1.5$, for t large enough, it follows that $\left(1 - \frac{1}{t}\right)^{(\alpha-1)t} \approx \exp(-1/2) \geq 1/2$. Thus, for t large enough and any integer $t' \in [t, 1.5t]$

$$X_{\lfloor 1.5t \rfloor} \geq \frac{1}{2} X_{t'} \tag{25}$$

Define $Y_k = X_{\lfloor 1.5^k \rfloor}$ for $k \geq 0$. Then, the following are direct implications of (25): for any $\theta > 0$,

$$\begin{aligned} X_\tau \geq \theta\tau \text{ i.o.} &\implies Y_k \geq \theta 1.5^k/3, \text{ i.o.}; \\ Y_k \geq \theta 1.5^k, \text{ i.o.} &\implies X_\tau \geq \theta\tau, \text{ i.o.}. \end{aligned} \tag{26}$$

The first implication is true because for any τ such that $X_\tau \geq \theta\tau$,we can find a k such that $\tau \in [1.5^{k-1}, 1.5^k]$. It then follows from (25) that $Y_k = X_{\lfloor 1.5^k \rfloor} \geq 1/2 X_\tau \geq \theta\tau/2 \geq \theta 1.5^{k-1}/2$. Similarly, for the second implication, whenever $Y_k \geq \theta 1.5^k$, taking $\tau = \lfloor 1.5^k \rfloor$, we can write $X_\tau \geq \theta 1.5^k \geq \theta\tau$.

It follows from (26) that $\Pr(X_\tau \geq \theta\tau) \leq \Pr\left(Y_k \geq \theta 1.5^k/3\right)$. Thus, in order to complete the proof of (18) by contradicting (24), and thereby (21), it is sufficient to show that for $\theta = \delta^2/(4N^2)$,

$$\Pr\left(Y_k \geq \theta 1.5^k/3, \text{ i.o.}\right) = 0.$$

For this, let event $E_k = \left\{Y_k \geq \theta 1.5^k/3\right\}$. Then, from $\mathbb{E}[Z_t] \leq O(1)$, relations $Y_k = X_{1.5^k}$, $Z_k = \frac{X_k}{\log k}$ and Markov's inequality we obtain that

$$\Pr(E_k) \leq \frac{3\mathbb{E}[Y_k]}{\theta 1.5^k} = \frac{3\log\lfloor 1.5^k \rfloor \mathbb{E}\left[Z_{\lfloor 1.5^k \rfloor}\right]}{} \leq O\left(\frac{k}{1.5^k}\right).$$

Therefore,

$$\sum_k \Pr(E_k) \leq \sum_k O\left(\frac{k}{1.5^k}\right) < \infty.$$

Therefore, by Borel-Cantelli's Lemma, we have that

$$\Pr(E_k \text{ i.o.}) = 0.$$

This completes the proof of the lemma. ■

Proof of Lemma 4: Define the following: for all n

$$\tilde{W}_n(\tau+1) = W_n(\tau) + 1 - \beta_n S_n^*(\tau)$$

where $S_n^*(\tau)$ is the schedule of MUCF algorithm and β_n is the inter-arrival time for the arrival process to queue n. When the queue is empty, treat β_n as an independent r.v. without any meaning, while if queue is not empty then treat it as the inter-arrival time between the packet being served and the packet behind it. In either case, due to the FIFO policy β_n is totally independent of the scheduling decisions performed by the algorithm till (and including) time τ and the information utilized by the algorithm. Therefore, we will treat it as an independent random variable with Geometric distribution of parameter λ_n (since arrival process is Bernoulli i.i.d.). Consider the following: for any τ,

$$F\left(\tilde{W}_n(\tau+1)\right) - F\left(W_n(\tau)\right) = \int_{W_n(\tau)}^{\tilde{W}_n(\tau+1)} f(x)dx \tag{27}$$

It is easy to see that,

$$\int_{W_n(\tau)}^{\tilde{W}_n(\tau+1)} f(x)dx = \int_0^{1-\beta_n S_n^*(\tau)} f\left(y + W_n(\tau)\right) dy \tag{28}$$

Since $f(\cdot)$ is non-decreasing bi-Lipschitz continuous with $f(0) = 0$, we have

$$\begin{aligned} f\left(y + W_n(\tau)\right) &= f\left(y + W_n(\tau)\right) - f\left(W_n(\tau)\right) + f\left(W_n(\tau)\right) \\ &\leq \left|f\left(y + W_n(\tau)\right) - f\left(W_n(\tau)\right)\right| + W_n^f(\tau) \\ &\leq \rho\,|y| + W_n^f(\tau) \end{aligned} \tag{29}$$

Now, it follows from (27), (28), and (29) that

$$\begin{aligned} &F\left(\tilde{W}_n(\tau+1)\right) - F\left(W_n(\tau)\right) \\ &\leq \rho\left(1 - \beta_n S_n^*(\tau)\right)^2 + \left(1 - \beta_n S_n^*(\tau)\right) W_n^f(\tau) \end{aligned} \tag{30}$$

Using (30) and the fact that β_n is a Geometric r.v. with mean $1/\lambda_n$, we have the following:

$$\begin{aligned} &\mathbb{E}\left[\sum_n \lambda_n F\left(\tilde{W}_n(\tau+1)\right) - \sum_n \lambda_n F\left(W_n(\tau)\right) \middle| W(\tau)\right] \\ &\leq \sum_n W_n^f(\tau)\lambda_n - \sum_n W_n^f(\tau) S_n^*(\tau) + \rho \sum_n \lambda_n \\ &\quad - 2\rho \sum_n S_n^*(\tau) + 2\rho \sum_n S_n^*(\tau)\lambda_n^{-1} \end{aligned} \tag{31}$$

Here we have used the fact that $S_n^*(\tau) \in \{0, 1\}$ and hence $\left(S_n^*(\tau)\right)^2 = S_n^*(\tau)$, and $\mathbb{E}\left[\beta_n^2\right] = 2/\lambda_n^2 - 1/\lambda_n \leq 2/\lambda_n^2$. Using

the fact that $\sum_n \lambda_n \leq N$, $\sum_n S_n^*(\tau) \leq N$ and $\lambda_n^{-1} < \infty$ for all n such that $\lambda_n \neq 0$, we obtain

$$\mathbb{E}\left[\sum_n \lambda_n F\Big(\tilde{W}_n(\tau+1)\Big) - \sum_n \lambda_n F\Big(W_n(\tau)\Big)\Big| W(\tau)\right] \leq \left\langle W_n^f(\tau), \lambda - S^*(\tau)\right\rangle + K \quad (32)$$

where K is a large enough constant. Now, define $S^w(\tau)$ as

$$S^w(\tau) = \arg\max_{\pi \in \mathscr{S}_{\max}} \sum_n \left\langle W^f(\tau), \pi\right\rangle.$$

That is, $S^w(\tau)$ is the maximum weight schedule with weight of queue n as $W_n^f(\tau)$. Consider the following:

$$\begin{aligned}&\left\langle W^f(\tau), \lambda - S^*(\tau)\right\rangle \\ &= \left\langle W^f(\tau), \lambda - S^w(\tau)\right\rangle + \left\langle W^f(\tau) - U^f(\tau), S^w(\tau) - S^*(\tau)\right\rangle \\ &\quad + \left\langle U^f(\tau), S^w(\tau) - S^*(\tau)\right\rangle. \end{aligned} \quad (33)$$

From the definition of $S^*(\tau)$, $S^w(\tau)$, $\Delta(\tau)(= W(\tau) - U(\tau))$, and bi-Lipschitz continuity of $f(\cdot)$ it follows that

$$\left\langle U^f(\tau), S^w(\tau) - S^*(\tau)\right\rangle \leq 0 \quad (34)$$

$$\left\langle W^f(\tau) - U^f(\tau), S^w(\tau) - S^*(\tau)\right\rangle \leq \rho \left\langle \Delta(\tau), \mathbf{1}\right\rangle \quad (35)$$

Now, for strictly admissible λ such that $\lambda = \sum_k \alpha_k \pi^k$ with $\sum_k \alpha_k = 1 - \gamma$ for some $\gamma \in (0,1)$, we obtain that

$$\begin{aligned}&\left\langle W^f(\tau), \lambda - S^w(\tau)\right\rangle \\ &= \left\langle W^f(\tau), \sum_k \alpha_k \pi^k\right\rangle - \left(\gamma + \sum_k \alpha_k\right)\left\langle W^f(\tau), S^w(\tau)\right\rangle \\ &= \sum_k \alpha_k \left\langle W^f(\tau), \pi^k - S^w(\tau)\right\rangle - \gamma \left\langle W^f(\tau), S^w(\tau)\right\rangle \end{aligned} \quad (36)$$

Since $S^w(\tau)$ is the maximum weight schedule with weight of queue n as $W_n(\tau)$:

$$\left\langle W^f, \pi^k - S^w(\tau)\right\rangle \leq 0 \quad \forall k \quad (37)$$

Thus, it follows from (36) and (37) that

$$\left\langle W^f(\tau), \lambda - S^w(\tau)\right\rangle \leq -\gamma \left\langle W^f(\tau), S^w(\tau)\right\rangle. \quad (38)$$

Now since all N entries can be covered by N distinct feasible schedules, it follows that the weight of maximum weight matching is at least $1/N$ the sum of weights of all the entries. That is

$$\begin{aligned}\left\langle W^f(\tau), S^w(\tau)\right\rangle &\geq \frac{1}{N}\sum_n W_n^f(\tau) \\ &= \frac{\left|W^f(\tau)\right|}{N} \geq \frac{1}{\rho}\frac{|W(\tau)|}{N}\end{aligned} \quad (39)$$

The last inequality follows from the bi-Lipschitz continuity of $f(\cdot)$. Combining (32)-(39) and taking further expectation with respect to $W(\tau)$, we obtain

$$\mathbb{E}\left[\sum_n \lambda_n F\Big(\tilde{W}_n(\tau+1)\Big) - \sum_n \lambda_n F\Big(W(\tau)\Big)\right] \leq \varepsilon \mathbb{E}\left[|W(\tau)|\right] + \rho \mathbb{E}\left[|\Delta(\tau)|\right] + K, \quad (40)$$

where $\varepsilon = \frac{\gamma}{\rho N}$. To complete the proof, note that if queue n is non-empty after service at time τ, then $\tilde{W}_n(\tau+1) = W_n(\tau+1)$. Else, $W_n(\tau+1) = 0$ and thus it follows from (9) that $F\Big(\tilde{W}_n(\tau+1)\Big) \geq 0 = F(0) = F\left(W_n(\tau+1)\right)$. This inequality along with (40) implies the desired claim of Lemma 4. ∎

Proof of Lemma 6: This result corresponds to a constraint-free network in which scheduling at different output queues is independent. Hence, we will prove the result for a single queue operating under a WCP and strictly admissible loading. We use the same notation, but with subscript m's dropped. For a single queue operating under a WCP and strictly admissible loading, busy cycle lengths form an i.i.d process i.e., B^k are i.i.d. We now look at the large deviation behavior of this process. For a particular k and time $t = 0$ starting from the beginning of busy cycle B^k, let $I(t)$ denote the cumulative arrival process during B^k. Now consider the event $B^k > t$. If the length of the busy cycle is greater than t, it implies that the queue has been non-empty up to time t. Further, since the service process is work conserving, it follows that there has been one departure every time slot and hence a total of t departures up to time t. Since the total number of departures cannot be more than the total number of arrivals, it follows that $I(t) > t$. Thus, we conclude that the event $B^k > t$ implies the event $I(t) > t$. For large enough t, we can now write

$$\Pr\left(B^k > t\right) \leq \Pr\left(I(t) - t > 0\right) \leq C\exp\left(-Dt\right) \quad (41)$$

where C and D are some non-negative constants. The last inequality follows from Chernoff bound, which can be used because arrivals happen according to a Bernoulli process. Let Θ denote the random variable $\max_{k \leq \tau} B^k$. Then, we have the following:

$$\begin{aligned}\mathbb{E}\left[\Theta\right] = \sum_t \Pr\left(\Theta > t\right) &= \sum_{t<\Gamma} \Pr\left(\Theta > t\right) + \sum_{t \geq \Gamma} \Pr\left(\Theta > t\right) \\ &\leq \Gamma + \sum_{t \geq \Gamma} \Pr\left(\Theta > t\right)\end{aligned} \quad (42)$$

(42) is true for any non-negative integer Γ. In particular, choose Γ large enough such that (41) is true $\forall t \geq \Gamma$. It now follows from union bound that

$$\begin{aligned}\sum_{t \geq \Gamma} \Pr\left(\Theta > t\right) &\leq \sum_{k \leq \tau}\sum_{t \geq \Gamma} \Pr\left(B^k > t\right) \\ &\leq O\left(\tau \exp(-D\Gamma)\right)\end{aligned} \quad (43)$$

The second inequality follows from (41). Now by choosing $\Gamma = O\left(\log \tau\right)$ we can bound $\sum_{t \geq \Gamma} \Pr\left(\Theta > t\right)$ by 1. It now follows from (43) that

$$\mathbb{E}\left[\max_{k \leq \tau} B^k\right] \leq O\left(\log \tau\right).$$

∎

VIII. Experiments

We carried out simulations to evaluate the performance of our algorithm in the context of IQ switches. We assumed a FIFO queuing policy at the input ports of the IQ switch. We compared the performance of our algorithm with the Longest Queue First (LQF) and Oldest Cell First (OCF) algorithms.

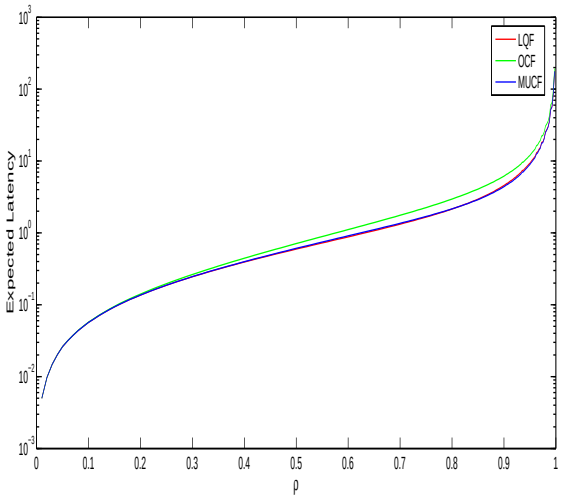

Figure 5. Comparison of the logarithm of Expected latencies of different scheduling algorithms

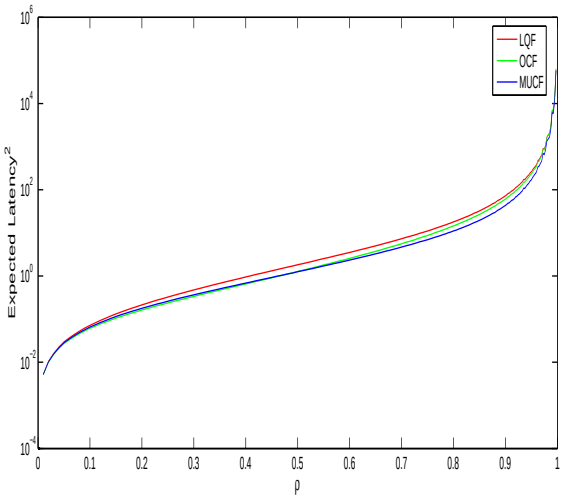

Figure 6. Comparison of the logarithm of second moments of the latencies of different scheduling algorithms.

We used the fixed-length packet switch simulator available at http://klamath.stanford.edu/tools/SIM/.

We first explain the simulation setting: The switch size is $N = 16$. The buffer sizes are infinite. The policy used is FIFO. All inputs are equally loaded on a normalized scale, and $\rho \in (0, 1)$ denotes the normalized load. The arrival process is Bernoulli i.i.d. We use a Uniform load matrix, i.e., $\lambda_{ij} = \rho/N \ \forall i, j$. We ran our simulation for 2.1 million time steps removing the first $100,000$ time steps to achieve steady-state.

Because we are approaching this problem from the perspective of fairness, we evaluate the aforementioned switching algorithms in terms of Latency and Output-Queue (OQ) Delay. OQ delay is defined as the difference of the departure times of a packet in the input queued switch and the shadow OQ switch. Further, the goal cannot only be to achieve a better expected latency, but in fact, we wish to value consistency, or relatively few deviations from the mean. One measure for this are higher moments of the variables. Thus, here we provide plots for the logarithm of first and second moments of both Latency and the OQ Delay versus a uniform load of ρ. Figures 5 and 6 plot respectively the logarithm of the first and second moments of the latency. We observe that for lower loads, i.e., for $\rho < 0.45$ the performance of all the three algorithms is almost the same. But for higher loads, the first moment of LQF and MUCF are better than OCF. Fig. 6 shows that in terms of the second moment, MUCF performs the best and LQF the worst, with OCF lying between. This is in line with our expectations because, as mentioned earlier LQF is not fair and hence performs badly at higher moments. MUCF performs better than OCF for both the moments.

Figures 5 and 6 correspond to latency and figures 7 and 8 correspond to OQ delay. We observe that MUCF performs better than the other two algorithms for both the metrics at all the loads, especially for the second moments illustrating fairness. Thus, the simulations illustrate that MUCF tracks the performance of an OQ switch better than LQF and OCF.

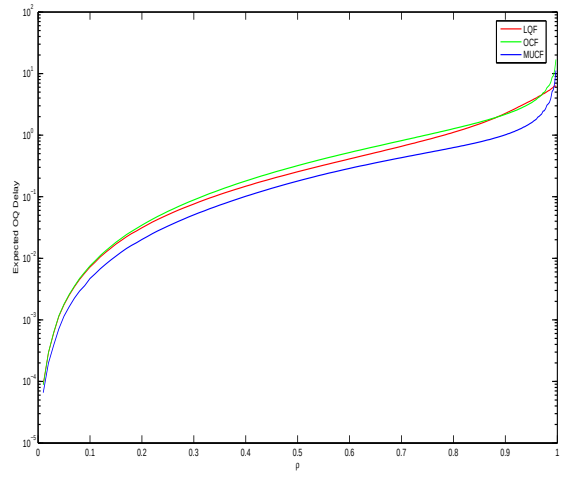

Figure 7. Comparison of the logarithm of expected output queued delays of different scheduling algorithms.

IX. Conclusion

In this paper, we considered the problem of designing a fair scheduling algorithm for constrained queuing systems. Fairness in networks is not only an intuitively desired goal, but also one with many practical benefits. Most of the existing work concentrates on fairly allocating bandwidth to different flows in the network. A major limitation of this approach is that it disregards the packetized nature of flows. We overcame this problem and proposed a packet based notion of fairness by establishing a novel analogy with the ranked election problem. Ranked election is a widely studied problem in the Economics literature, and this analogy allowed us to leverage that work. This results in a packet based notion of fairness and an algorithm to achieve this fairness.

Rather surprisingly, the algorithm turned out be the familiar MW style algorithm. Moreover, it does not require the knowledge of flow arrival rates. Our fairness algorithm

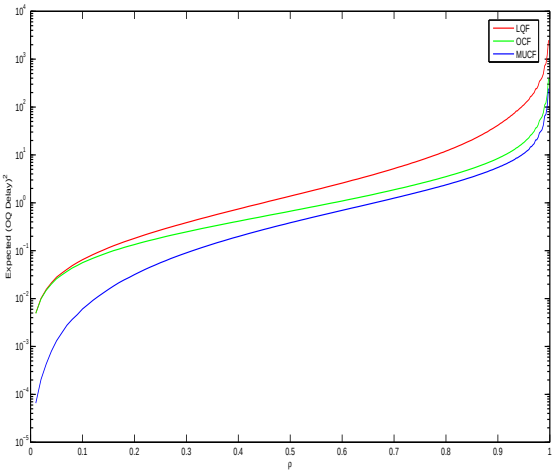

Figure 8. Comparison of the logarithm of second moments of the output queued delays of different scheduling algorithms.

also fits into the utility maximization framework that is more popular for designing fair algorithms. This, in some sense, validates our approach. We also proved that our algorithm is throughput optimal. This result is very crucial since the emulation approach already achieves fairness, but with a loss of throughput. Also, the proof is non-trivial and requires some non-traditional techniques to be introduced because existing proof techniques don't directly apply. We believe that the proof techniques we introduced are more widely applicable to similar problems and this is another important contribution of the paper. Finally, our simulation results corroborate the fact that our algorithm is better at providing fairness than the more popular algorithms in the context of input queued switches.

REFERENCES

[1] S. Keshav, *An engineering approach to computer networking: ATM networks, the Internet, and the telephone network*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[2] T. Bonald and L. Massoulié, "Impact of fairness on Internet performance," *Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 82–91, 2001.

[3] 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. Netw.*, vol. 13, no. 1, pp. 29–42, 2005.

[4] F. Kelly, A. Maulloo, and D. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, no. 3, pp. 237–252, 1998.

[5] L. A. Goodman and H. Markowitz, "Social welfare functions based on individual rankings," *The American Journal of Sociology*, vol. 58, pp. 257–262, Nov. 1952.

[6] J. Nagle, "On packet switches with infinite storage," *Communications, IEEE Transactions on [legacy, pre - 1988]*, vol. 35, no. 4, pp. 435–438, Apr 1987.

[7] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *SIGCOMM '89: Symposium proceedings on Communications architectures & protocols*, (New York, NY, USA), pp. 1–12, ACM, 1989.

[8] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control inintegrated services networks: the single-node case," *Networking, IEEE/ACM Transactions on*, vol. 1, no. 3, pp. 344–357, 1993.

[9] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control inintegrated services networks: the multiple node case," *Networking, IEEE/ACM Transactions on*, vol. 2, no. 2, pp. 137–150, 1994.

[10] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round-robin," *IEEE/ACM Transactions on Networking (TON)*, vol. 4, no. 3, pp. 375–385, 1996.

[11] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, no. 4, pp. 397–413, 1993.

[12] R. Pan, B. Prabhakar, and K. Psounis, "Choke: a stateless aqm scheme for approximating fair bandwidth allocation," in *IEEE Infocom*, 2000.

[13] R. Pan, B. Prabhakar, L. Breslau, and S. Shenker, "Approximate fair allocation of link bandwidth," *IEEE Micro*, vol. 23, no. 1, pp. 36–43, 2003.

[14] S. H. Low, "A duality model of tcp and queue management algorithms," *IEEE/ACM Trans. on Networking*, vol. 11, no. 4, pp. 525–536, 2003.

[15] M. Chiang, S. Low, A. Calderbank, and J. Doyle, "Layering as optimization decomposition: A mathematical theory of network architectures," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 255–312, Jan. 2007.

[16] R. Srikant, *The Mathematics of Internet Congestion Control*. Birkhäuser, 2004.

[17] G. de Veciana, T. Konstantopoulos, and T. Lee, "Stability and performance analysis of networks supporting elastic services," *IEEE/ACM Transactions on Networking (TON)*, vol. 9, no. 1, pp. 2–14, 2001.

[18] B. Prabhakar and N. McKeown, "On the speedup required for combined input and output queued switching," *Automatica*, vol. 35, no. 12, pp. 1909–1920, 1999.

[19] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching output queueing with a combined input output queued switch," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1030–1039, 1999.

[20] X. Zhang and L. Bhuyan, "Deficit Round-Robin Scheduling for Input-Queued Switches," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 4, pp. 584–594, May 2003.

[21] S. Li, J.-G. Chen, and N. Ansari, "Fair queueing for input-buffered switches with back pressure," *ATM, 1998. ICATM-98., 1998 1st IEEE International Conference on*, pp. 252–259, 22-24 Jun 1998.

[22] N. Ni and L. N. Bhuyan, "Fair scheduling in internet routers," *IEEE Trans. Comput.*, vol. 51, no. 6, pp. 686–701, 2002.

[23] M. Hosaagrahara and H. Sethu, "Max-Min Fairness in Input-Queued Switches," *ACM SIGCOMM poster session, Philadelphia, PA, USA, August*, 2005.

[24] D. P. Y. Yang, "Max-min fair bandwidth allocation algorithms for packet switches," *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1–10, 26-30 March 2007.

[25] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *Automatic Control, IEEE Transactions on*, vol. 37, no. 12, pp. 1936–1948, Dec 1992.

[26] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *Communications, IEEE Transactions on*, vol. 47, no. 8, pp. 1260–1267, 1999.

[27] D. Shah and D. Wischik, "Optimal scheduling algorithms for input-queued switches," in *IEEE Infocom*, 2006.

[28] N. Kumar, R. Pan, and D. Shah, "Fair scheduling in input-queued switches under inadmissible traffic," in *IEEE Globecom*, 2004.

[29] K. Arrow, *Social Choice and Individual Values*. Yale University Press, 1951.

Srikanth Jagabathula Srikanth Jagabathula received the BTech degree in Electrical Engineering from the Indian Institute of Technology (IIT) Bombay in 2006, and the MS degree in Electrical Engineering and Computer Science from the Massachusetts Institute of Technology (MIT), Cambridge, MA in 2008. He is currently a doctoral student in the Department of Electrical Engineering and Computer Science at MIT. His research interests are in the areas of revenue managemenr, choice modeling, queuing systems, and compressed sensing. He received the President of India Gold Medal from IIT Bombay in 2006. He was also awarded the "Best Student Paper Award" at NIPS 2008 conference and the Ernst Guillemin award for the best EE SM Thesis.

Devavrat Shah Devavrat Shah is currently a Jamieson career development associate professor with the department of electrical engineering and computer science, MIT. He is a member of the Laboratory of Information and Decision Systems (LIDS) and affiliated with the Operations Research Center (ORC). His research focus is on theory of large complex networks which includes network algorithms, stochastic networks, network information theory and large scale statistical inference. He received his BTech degree in Computer Science and Engg. from IIT-Bombay in 1999 with the honor of the President of India Gold Medal. He received his Ph.D. from the Computer Science department, Stanford University in October 2004. He was a post-doc in the Statistics department at Stanford in 2004-05. He was co-awarded the best paper awards at the IEEE INFOCOM '04, ACM SIGMETRICS/Performance '06; and best student paper awards at Neural Information Processing Systems '08 and ACM SIGMETRICS/Performance '09. He received 2005 George B. Dantzig best disseration award from the INFORMS. He received the first ACM SIGMETRICS Rising Star Award 2008 for his work on network scheduling algorithms.