

# Max-Min Fair Self-Randomized Scheduler for Input-Buffered Switches

Vahid Tabatabaee, Leandros Tassiulas

Department of Electrical and Computer Engineering and Institute for Systems Research

University of Maryland at College Park

emails: vahid@eng.umd.edu, leandros@eng.umd.edu

**Abstract**—In this paper, we consider self-randomized scheduling policies for input buffered switches. We provide a general architecture for the design of self-randomized algorithms. The common trend in the design of self-randomized schedulers is to use the number of backlogged cells as the weight function and to use instantaneous cell arrival graph to generate a candidate matching for scheduling. We discuss some of the shortcomings of this approach, and introduce the total arrival graph as an alternative for instantaneous arrival graph to improve the performance. We also introduce the concept of maxmin fair self-randomized scheduling algorithms. The idea here is to introduce self-randomized algorithms that can provide QoS by sharing the switch bandwidth proportional to some assigned weights. In order to study and compare the performance of the proposed scheduling algorithms, several simulations are carried out and their results are provided and discussed.

## I. INTRODUCTION

Input-buffered switch architecture has become the dominant architecture for high speed switch fabrics. The main reason for popularity of input-buffered switches is that their memory bandwidth requirements is lower than the alternative architectures, such as output-buffered and shared memory architectures. Moreover, the memory speed requirements of input-buffered switches is independent of the number of ports, where as for other architectures memory speed grows linearly with number of ports. Therefore, input-buffered architecture is scalable both in terms of line speed and port numbers.

In the input-buffered architecture, arriving cells are buffered in input buffers. At each input port of an  $N \times N$  switch there are  $N$  separate queues, which are called virtual output queues (VOQ). Each VOQ is associated to one of the outputs and cells are buffered based on their destination ports. Introduction of VOQs resolves head of line (HOL) blocking in the input-buffered switches [1]. In fact, without VOQs, throughput of input-buffered switches is limited to 58.6% [8].

Scheduler determines performance and characteristics of input-buffered switch fabrics. At every time slot, scheduler figures out which input should be connected to which output. This is basically equivalent of finding a matching in a bipartite graph, where the input ports and output ports are two separate set of graph vertices and there is an edge between a pair of input/output vertices if there is a request to send a cell between them. In a terabit switch, scheduler only has a few clock cycles to determine the matching for each time slot, which makes it even harder to design an efficient scheduler.

Throughput of the switch is the first performance measure that should be taken into account. Most of the initial theoretical studies focuses on finding 100% throughput schedulers. For

instance, the papers [9], [12], [4] prove that if the incoming traffic is admissible, maximum weighted matching (MWM) achieves 100% throughput. Weight of a link is usually considered to be number of buffered cells in the corresponding VOQ. The incoming traffic is called admissible if total arrival rate of every input and output port is less than one.

Deterministic sequential algorithms such as MWM are too complex for practical implementation in high speed switches. More recently, a new class of randomized scheduling algorithms are proposed [13]. These algorithms have low complexity, and at the same time achieve 100% throughput. The basic idea is instead of deriving the scheduling matching, to select it from a set of predetermined candidates. A simple randomized procedure is used to obtain or update the candidate set in every time slot. A completely randomized set results in a very low performance system. Tassiulas introduced the first stable (100% throughput) randomized scheme with linear complexity [13]. The candidate matching set in his scheme contains two matchings, one randomly selected matching and the matching that was used in the previous time slot. The candidate matching with higher total weight of links is selected, where weight of the link is number of backlogged cells.

Although this randomized algorithm is stable, it does not have good delay performance. Paper [5] proposed several modifications to the basic randomized algorithm to improve its delay performance. One of the contributions of [5] is introduction of the self-randomized algorithms. The idea behind self-randomized scheduling algorithms is instead of selecting a pure random matching, using the cell arrival pattern to generate a candidate matching for the scheduling. Even though the proposed self-randomized matchings have better delay performance, they still suffer from serious short comings as we discuss and illustrate in this paper.

Self-randomized schedulers rely on the arrival pattern as the source of generating good candidate matchings. At the lower arrival rates, since cell arrival occurs very rarely, the generated candidate matchings turn to be very sparse with very few links. Hence, they are not really good and efficient candidates for scheduling. To circumvent this problem, we propose to use total cell arrival graph instead of the instantaneous arrival graph to generate the candidate matching.

QoS provisioning is the other performance measure for scheduling algorithms that is overlooked in the context of randomized scheduling algorithms. Rate aware scheduling algorithms are very common for scheduling in output-buffered switches, however for the input-buffered schedulers with no speedup problem is more challenging, and there are few

theoretical and practical results [2], [10]. Paper [14] introduces and studies a new token based max-min fair scheduling algorithm. The basic idea behind a max-min fair scheduler is to allocate bandwidth among flows proportional to their weights, and if a flow can not utilize its bandwidth, because of constraint elsewhere in the network, then the residual bandwidth is distributed proportionally among others [6]. Note that this scheduling algorithm is not originally proposed for input-buffered switches, nevertheless it is directly applicable to them too. We can categorize this scheduling algorithm as a *deterministic* MWM scheduling algorithm, where weight of the links (flows) are number of tokens allocated to them. Tokens are distributed among links with their max-min rate.

We have extended the idea of maxmin fair scheduling algorithms and introduce the concept of maxmin fair self-randomized scheduling algorithms in this paper. We will use both the token generation process and the arrival process for generating the candidate matching. Furthermore, in our proposed scheme, weight of the links is a function of number of tokens and the number of backlogged cells. In this way, the proposed algorithm not only provides rate guarantees, but has even better average delay performance compared to the previous self-randomized algorithms.

In this paper, we have intentionally focused on the practical results of our work for self-randomized schedulers. We have also been able to generalize the previous stability results of [13]. More specifically, we have used fluid model techniques to show that the self-randomized schedulers deliver 100% throughput for general arrival process that satisfy strong law of large numbers such that no input or output is oversubscribed. The interested reader is referred to [11].

The rest of the paper is organized as follows. Section 2 reviews some basic randomized scheduling algorithms and introduces the general architecture that is adopted for self-randomized schedulers in this paper. Section 3 introduces the idea of total arrival graph versus instantaneous arrival graph for generation of the candidate matchings. Section 4 applies the notion of max-min fair rate scheduling to introduce a rate-aware self-randomized scheduler. Section 5 provides simulation result for the proposed scheduling algorithms.

## II. MODELS AND DEFINITIONS

We consider input queued switches that serve fixed size cells. Each input and output has the capacity of serving 1 cell per unit time. Since queues exist only at the input ports, the latter assumption implies that traffic of at most 1 cell per unit time can be transferred from the input ports to a given output port. The scheduling policy is a matching algorithm  $m$  that based on the state of the switch selects a matching between the inputs and outputs in every time slot. If input  $i$  is matched to output  $j$ , and the corresponding VOQ is not empty, a cell is sent from input  $i$  to output  $j$ . A matching can be represented by a permutation matrix  $\pi$ . Input ports are represented by the rows and output ports by the columns of this matrix, therefore input  $i$  is matched to output  $j$  if and only if  $\pi_{ij} = 1$ .

Consider the following randomized scheduling algorithm,

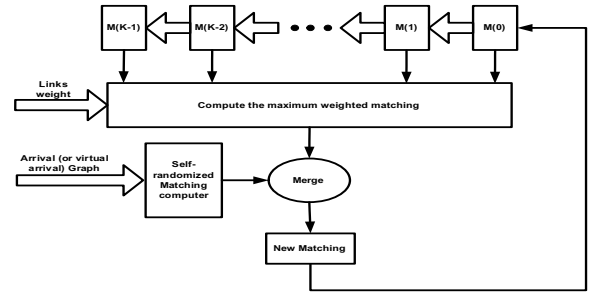


Fig. 1. The generic block diagram adopted for self-randomized schedulers.

### Algorithm 1: (Tass)[13]

- Let  $S(n)$  be the schedule used at time  $n$ .
- At time  $n + 1$  choose a matching  $R(n + 1)$  uniformly at random from the set of all permutation matrices  $\pi \in \Pi$ .
- $S(n + 1) = \arg \max_{\pi \in \{S(n), R(n+1)\}} \langle S, Z(n + 1) \rangle$

At every time slot, a random matching is generated, and its weight is compared to the weight of the matching that is used in previous time slot. The matching with highest weight is selected for scheduling in the present time slot. Paper [13] proves that the above algorithm is stable under any Bernoulli i.i.d. admissible input. This proof is the basis for all subsequent stability proofs for other randomized scheduling algorithms.

Although this randomized algorithm is stable, it does not have a good delay performance. Paper [5] proposed several modifications to the basic randomized algorithm to improve its delay performance. One of the contributions of [5] is introduction of the self-randomized algorithms. The idea behind self-randomized scheduling algorithms is instead of selecting a pure random matching, using the arrival cell pattern as the source of randomization. SERENA is the self-randomized algorithm introduced in [5]:

### Algorithm 2: SERENA

- Let  $S(t - 1)$  be the schedule (matching) used at time  $t - 1$ .
- Let  $A(t) = [a_{ij}(t)]_{N \times N}$  denote the arrival graph, where  $a_{ij}(t) = 1$  indicates arrival, and  $a_{ij}(t) = 0$  indicates no arrival at time  $t$ .
- Let the schedule at time  $t$ ,  $S(t)$ , be the matching that results from merging  $A(t)$  and  $S(t - 1)$ .

The merging process selects a combination of the two input matching links and generate a matching with higher weight than the original matchings.

Next, we introduce the general architecture that we will use in this paper for self-randomized schedulers. The scheduler structure is shown in figure 1 and it can be divided into two stages. First stage has a memory bank that buffers  $K$  previously used matchings. The first stage also contains the circuitry that selects the candidate matching with highest weight out of the  $K$  matchings. The candidate matching is passed to the second stage.

Randomized scheduling algorithms usually store and process the matching that is used in the previous time slot; we have extended this idea by taking into account matchings that are used in  $K$  previous time slots. Memory or state notion

plays a critical role in the stability and performance of the randomized schedulers. Saving the previous matching and using it as a new candidate is the main reason for the stability of Tassiulas algorithm, and for that reason it has been retained in all proposed self-randomized algorithms.

The concept of maintaining  $K$  previous matchings is also in accordance with some of the proposed deterministic schemes. It is proved in [2] that fixed serving rates can be obtained, using a periodic sequence of matching matrices for scheduling. The rate matrix is decomposed to a sequence of permutation matrices, that should be repeated in a periodic form. Based on the same result, paper [7] has proposed a rate provisioning scheme that uses a weighted fair queueing scheduler to select one matching from a fixed set of matchings. Increasing the number of stored matchings enables us to potentially reuse the same set of matchings, and improve the performance, specially if we intend to do rate provisioning.

Now we proceed to the second stage of the scheduler in figure 1. The Main functions of the second stage are, to generate a candidate matching in the self-randomized block, and to merge that matching with the candidate matching out of the first stage. The end result of the merging block is the scheduling matching for that time slot. This matching is also passed to the first stage and is stored in the memory bank for future use in subsequent time slots.

The performance of the scheduler heavily depends on the procedure that is used to generate the self-randomized matching. Basically, a self-randomized graph is generated based on the cell arrival and/or token generation process as we discuss later. Then, the self-randomized matching is extracted from the self-randomized graph simply by eliminating some of the contending links in the self-randomized graph. For the elimination, first we consider the contending links at the output port and for each output port we keep the link with highest weight. Then, we repeat this procedure for the input port ports considering only those links that are prevailed the output port contention.

In this architecture, scheduling encompasses weight calculation for  $K$  matchings at the first stage, and one merging at the second stage. This structure is based on the fact that weight of a matching can be simply calculated in hardware, where as merging of two matchings is not a simple scalable task for hardware.

Basically, the algorithms that are introduced in this paper are specified by the weight function and the self-randomized graph generator procedure. The weight function specifies the fundamental features and objectives of the algorithm, while the self-randomized matching generator determines how well and fast scheduler responds and adapts to the changes in the arrival pattern.

### III. INSTANTANEOUS VERSUS TOTAL ARRIVAL FOR SELF-RANDOMIZATION

In the self-randomized schedulers such as SERENA the arrival pattern at every time slot is used to generate the candidate matching for that time slot. In other words, the

candidate matching is based on the instantaneous arrivals. The weight function is simply number of backlogged cells.

In this approach, every new arrival gets one chance to be included in the self-randomized matching. However, links may lose their chance, either to the other new arrivals in the process of generating the self-randomized matching, or in the merging phase. If a link loses its chance, it should wait for another new arrival to get another opportunity. Therefore, counter-intuitively, under low utilization, it is possible that some of the cells incur high delays. In fact, the situation is worse for very low arrival rates, since the average waiting time for the next arrival is higher. We intend to resolve this undesirable issue.

To resolve this problem we can use the total arrival graph as the source for generation of the self-randomized candidate matching. In the total arrival graph, there is a link for all previous cell arrivals that are not scheduled yet. Total arrival graph evolution is based on link insertion, and ejection rules in every cell time.

- **Insertion** : Any link with new arrival that is not in the total arrival graph is added to the total arrival graph.
- **Ejection** : Links that are scheduled are removed from the total arrival graph.

At every cell time a matching is extracted from the total arrival graph. The extraction process is a trivial extension of the process that is used in SERENA. Similar to SERENA, for every output port the link with largest weight is selected from the total arrival graph. However, there may be some contention at the input ports between the selected links too. In the next step, if there are multiple selected links connected to the same input port, the one with highest weight is selected. The outcome of this two step selection process is a matching that is used as the self-randomized matching.

In the simulation section, we demonstrate that this simple modification improves the performance of the schedulers.

### IV. MAX-MIN FAIR BASED SELF-RANDOMIZED SCHEDULER

In this section, we introduce a new class of self-randomized scheduling algorithms that use a set of token values for the link weight function. Token based weight function enables us to do bandwidth allocation among different flows based on their reserved rates. In the context of prioritized max-min fair scheduling algorithms each flow  $i$  has a normalized reserved rate or priority,  $w_i$ . Greater the reserved rate of a flow, better the service it should receive, subject to bandwidth availability. A bandwidth allocation is max-min fair if it is not possible to increase the bandwidth of any flow  $i$ , without hurting another flow  $j$  for which  $g_j(n)/w_j \leq g_i(n)/w_i$ , where  $g_i(n)$  is the bandwidth allocated to flow  $i$  at time  $n$ .

Tassiulas and Sarkar [14] have proposed a token based deterministic max-min fair scheduling algorithm that works based on the MWM algorithm. Weight of every flow depends on an estimate of the max-min fair bandwidth of the flow, previous service received by the flow, and traffic demand of the flow. The proposed algorithm achieves max-min fair objective

for any arbitrary set of flows and graphs. More specifically, their result is general and is valid for an arbitrary network graph topology and a set of flows traversing arbitrary paths in the network. We will use and extend the same concepts in the context of self-randomized scheduling algorithms, and introduce a new class of randomized scheduling algorithms that have rate provisioning capabilities for input buffered switches.

In the original algorithm, at every node, a WRR scheduler distributes tokens between eligible flows. A flow is eligible to receive a token if, 1) it has less token than the number of backlogged cells (queue condition) and 2) number of tokens allocated to a flow at each end does not exceed the number at the other end by more than a constant  $W$  (token condition). Note that each end node of a link allocates token to the link; weight of a link is minimum number of allocated tokens to it. When a link is scheduled, one token is taken out of its allocated token buckets.

In our scheme, we have removed the queue condition, and consequently number of tokens allocated to a flow can exceed the number of backlogged cells. Weight of a link is minimum of number of tokens allocated to it in its end nodes and number of backlogged cells plus a constant  $B$ . In this way, even though the number of tokens is not limited by the number of backlogged cells, the weight of a link can not exceed number of backlogged cells by more than a constant  $B$ . In this way, once a cell arrives it does not necessarily need to wait for the scheduler to give it a token if there are already some extra tokens in the bucket.

The self-randomized graph link insertion and ejection rule is based on the increase of the link weight and cell arrival as it is explained below:

- **Insertion** : Any link with new arrival or increase in the weight that is not in the total arrival graph is added to the total arrival graph.
- **Ejection** : Links that are scheduled are removed from the total arrival graph.

In this way, the links with higher weight have a higher chance to be scheduled.

## V. SIMULATION RESULTS

In this section, we use simulation results to study and discuss the delay performance of the introduced self-randomized scheduling algorithms. Simulations are done for a 16x16 input-buffered switch. The arrival process for all links are i.i.d Bernoulli, and the arrival rate for the link between input  $i$  and output  $j$  is  $\lambda_{ij}$ .

Two different load distribution models, uniform, and log-diagonal between the links are studied. For the uniform, the total load  $\rho$  of every input port is uniformly distributed among its outgoing links. In the log-diagonal model  $\lambda_{i|j|} = 2\lambda_{i|j+1|}$ ,  $i = 1 \dots, N, j = i, \dots, N + i$ . We have also simulated diagonal traffic model [5] and observed the same general trend, but would not report it here for the sake of brevity.

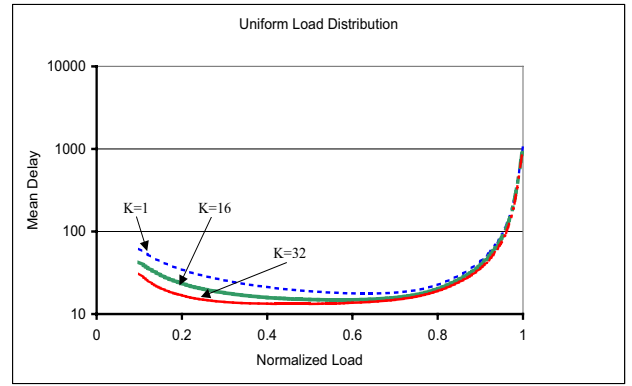


Fig. 2. Average delay versus load for different memory sizes.

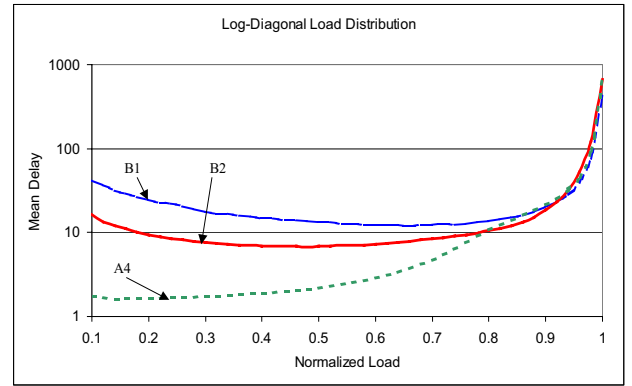


Fig. 3. Mean delay for three schemes based: B1 (instantaneous arrival), B2 (total arrival) and A4 (max-min fair rate) under log-diagonal traffic.

In the first we study the effect of  $K$ , memory length for the buffered matchings. We consider a self-randomized scheduler that uses number of backlogged cells as the weight function and total cell arrival to generate the self-randomized graph. The memory length for the buffered matchings  $K$  are set to 1, 16, and 32. The results for uniform traffic distribution model is shown in Fig. 2, where the average delay versus traffic load is plotted. Advantage of employing memory for the matching is considerable, and delay performance is evidently enhanced by increasing the memory size from 1 to 16 and 16 to 32. For the rest of the experiments, we fix  $K$  to 32.

In the next set of experiments we seek to compare the average delay performance of the proposed scheduling algorithms for different traffic distributions. Three different scheduling algorithms are studied. The first one (B1) is very similar to the SERENA [5]. The only main difference is in  $K$  where it is set to 32 instead of 1. The second algorithm is B2 that works based on total cell arrival graph as is explained in section III. The third algorithm A4 is the max-min fair self-randomized scheduling algorithm that explained in section IV. Figures 3 and 4 shows the result for log-diagonal and uniform distributions respectively.

An interesting and non-desirable phenomena is having large delays at very low rates for B1. This is to some extent,

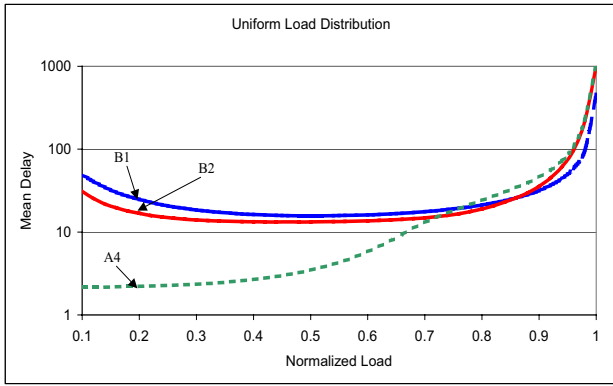


Fig. 4. Mean delay for three schemes based: B1 (instantaneous arrival), B2 (total arrival) and A4 (max-min fair rate) under uniform traffic.

because of the self-randomized matching generator that is used. Note that new links are introduced to the system through the self-randomized graph, and a link is introduced to the self-randomized graph, only when there is a new arrival for that link. Therefore, if a link is not included in the self-randomized matching after its first cell arrival, it should wait for the next cell arrival to get another chance. This behavior is not clear in [5] at the first glance, since authors have reported and plotted average queue length, rather than the average delay in their simulation results.

In B2, the performance at low rates is better than B1, but we still observe a minimum in the average delay plots. Recall that the self-randomized graph in B2 is based on the total cell arrival graph and a link is introduced to the candidate matching until it is scheduled. Nevertheless, at the very low rate the cell arrival graph is not able to provide good candidate matching.

This problem is completely resolved in the A4 algorithm. In the A4, the links are inserted to the self-randomized graph when there is an arrival or there is an increase in the link weight. If the weight of a link is less than the number of backlogged cells plus  $B$ , it grows with the max-min rate. The max-min rate is much higher than the arrival rate if the arrival rate is low.

In the last experiment, we study the max-min fairness behavior of the proposed algorithm. We consider a 4x4 input-buffered switch fabric, with log-diagonal weight matrix. The average serving rate for all links are calculated for every 1000 time slots, and plotted.

We study how well the algorithm tracks the max-min fair rate, and how fast it adapts the serving rate to the changes in arrival rate. We start the experiment with all queues backlogged, then in the middle of simulation we stop arrivals for links (1, 1), and (2, 2). The max-min fair rates, before and after the shut down are given below in rate matrices  $R_1$  and  $R_2$  respectively,

$$R_1 = \begin{pmatrix} 0.533 & 0.267 & 0.133 & 0.067 \\ 0.067 & 0.533 & 0.267 & 0.133 \\ 0.133 & 0.067 & 0.533 & 0.267 \\ 0.267 & 0.133 & 0.067 & 0.533 \end{pmatrix} \quad R_2 = \begin{pmatrix} 0. & 0.8 & 0.133 & 0.067 \\ 0.6 & 0. & 0.267 & 0.133 \\ 0.133 & 0.067 & 0.533 & 0.267 \\ 0.267 & 0.133 & 0.067 & 0.533 \end{pmatrix}$$

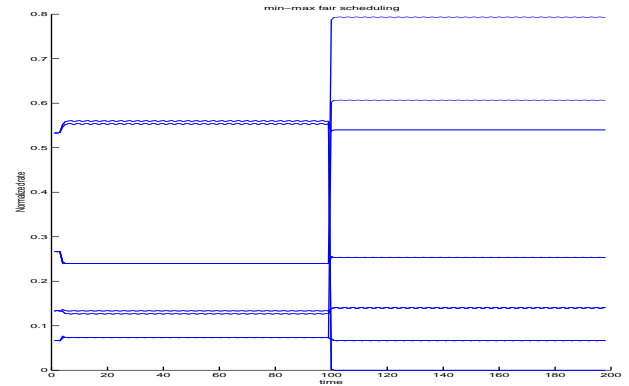


Fig. 5. Serving rate adaptation to arrival pattern changes for the max-min fair self-randomized scheduler. The arrival rate is changed at time 100.

Simulation results are given in Fig. 5. The arrival pattern changes at time frame 100 in the plot. Each time frame consists of 1000 cell times. The serving rate adapts to the new arrival pattern immediately. This experiment illustrates that serving rate of A4 scheme is very close to the maxmin rates. Moreover, algorithm adapts to the changes of arrival pattern very rapidly.

## REFERENCES

- [1] T. Anderson, S. Owicki, J. Saxe, C. Thacker, "High Speed Switch Scheduling for Local Area Networks", *ACM Transactions on Computer Systems*, pp. 319-352, Nov 1993.
- [2] C.S. Chang, W.J. Chen, H.Y. Huang, "Birkhoff-von Neuman Input-Buffered Crossbar Switches", *INFOCOM '00*
- [3] J.G. Dai, "Stability of fluid and stochastic processing networks", Miscellanea Publication, No.9, Center for Mathematical Physics and Stochastics, Denmark, (<http://www.maphysto.dk>), January 1999.
- [4] J.G. Dai, B. Prabhakar, "The throughput of data switches with and without speedup", *INFOCOM '00*
- [5] P. Giaccone, B. Prabhakar, D. Shah, "Towards simple, high-performance schedulers for high-aggregate bandwidth switches", *INFOCOM '02*
- [6] E.L. Hahne, "Round-robin scheduling for max-min fairness in data networks", *IEEE Journal on selected areas in communications*, v. 9, no. 7, pp. 1024-1039, Sep. 1991.
- [7] A. Hung, G. Kesidis, N. McKeown, "ATM input-buffered switches with guaranteed rate property", *Proc. IEEE ISCC'98*, Athens, pp. 331-335, 1998.
- [8] M.J. Karol, M.G. Hluchyj, S.P. Morgan, "Input Versus Output Queueing on a Space-Division Packet Switch", *IEEE Transactions on Communications*, v.35, pp. 1347-1356, 1987
- [9] N. McKeown, V. Anantharam, J. Warland, "Achieving 100% Throughput in an Input-Queued Switch", *INFOCOM '96*, pp. 296-302.
- [10] V. Tabatabaee, L. Georgiadis, L. Tassiulas, "QoS Provisioning and Tracking Fluid Policies in Input Queueing Switches", *IEEE Transaction on Networking*, v.9, no.5, pp. 605-617, Oct. 2001.
- [11] V. Tabatabaee, "Scheduling and Rate Provisioning for Input-Buffered Cell Based Switch Fabrics", PhD Dissertation, Department of Electrical and Computer Engineering, University of Maryland at College Park, 2003.
- [12] L. Tassiulas, A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks", *IEEE Trans. on Automatic Control*, v.37, n.12, Dec. 1992, pp. 1936-1948.
- [13] L. Tassiulas, "Linear complexity algorithms for maximum throughput in radio networks and input queued switches", *IEEE INFOCOM'98*, v.2, New York, 1998, pp. 533-539.
- [14] L. Tassiulas, S. Sarkar, "Maxmin fair scheduling in wireless networks" *IEEE INFOCOM'02*.