This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2007 proceedings.

Efficient, fully local algorithms for CIOQ switches

Amin Firoozshahian, Vahideh Manshadi, Ashish Goel, Balaji Prabhakar
Stanford University
Email: {aminf13, vahidehh, ashishg, balaji}@stanford.edu

***Abstract*— A number of algorithms have been proposed in the literature for scheduling CIOQ switches. The algorithms which have been proven to provide strict performance guarantees on delay (via the emulation of an output-queued switch) have been too complicated to implement because they require the exchange of a large amount of information between inputs and outputs. With implementation as our primary focus, we consider scheduling algorithms that are "fully local." This means inputs and outputs must be able to make decisions regarding matchings using only local information (except requests, grants and accepts). This constraint, which is essentially necessary for high-speed implementations, appears too restrictive for designing algorithms which enable the emulation of an output-queued switch. Rather surprisingly, we find a very simple and fully local algorithm FLGS (for fully local Gale-Shapley) which, at a speedup of 2, emulates an output-queued switch implementing a number of different output link scheduling algorithms such as weighted round robin and strict priority. We explore the performance of the algorithm at speedups between 1 and 2 using simulations and find that it partitions the bandwidth nearly as well as an output-queued switch at speedups 1.2 or higher.**

I. Introduction

Switching theory and practice have developed tremendously during the past decade. In order to adequately motivate our work, it is useful to survey this development and consider the influence theory and practice have had on each other. The theory consists, in the main, of designing scheduling algorithms for a variety of switch architectures and developing tools for proving performance guarantees. Particularly, there are two main research threads for crossbar fabrics, which are of practical interest, depending on the fabric speedup: the focus at speedup 1 has been on algorithms which deliver 100% throughput and low average delays, and at speedup 2 it has been on algorithms which provide precise delay guarantees.

At speedup 1, the seminal result of Tassiulas and Ephremides [12] and McKeown et al [10] is that the maximum weight matching (MWM) algorithm delivers 100% throughput. Despite this highly desirable property, the MWM is not commercially implemented because it lacks the following two properties: (i) RGA: the request-grant-accept (RGA) mechanism of matching inputs and outputs, and (ii) FL: inputs and outputs operate fully locally (FL). The MWM algorithm and its subsequent variations ([12] and [7])are not amenable to an RGA implementation and are not FL because a (centralized) scheduler needs to know the backlogs (the weights) to find the MWM. On the other hand, the appeal of the scheduling algorithm iSLIP [9] derives from its possession of the RGA and the FL properties. Yet, despite being widely-used ([14]), no guarantees of throughput or delay can be made about iSLIP at speedup 1[1]. Thus, theorists face the following still-open question: *Does there exist a switch scheduling algorithm which possesses the RGA and the FL properties which delivers 100% throughput at speedup 1?*

For speedups larger than one, Prabhakar and McKeown [11] obtained an algorithm for a CIOQ (combined input- and output-queued) switch running at speedup 4 which allows it to *exactly emulate* an OQ switch. Exact emulation means under identical inputs the CIOQ switch will match, cell-by-cell, the departures from an OQ switch. Chuang et. al. [4] then showed that speedup of 2-1/N is necessary and sufficient for such emulation. However, the amount of information to be exchanged between inputs and outputs in these algorithms is prohibitively large; for example, time-stamp information, flow IDs, etc. Other algorithms proposed by Charny et. al. [3] and Krishna et. al. [8] also do not possess the FL property. This strand of research, therefore, culminated in the impression that the following question has a negative answer: *Is there a switch scheduling algorithm which possesses the RGA and the FL properties and enables the emulation of an OQ switch at speedup 2?*

Our main contribution is an RGA algorithm which possesses the FL property, called FLGS (for fully local Gale-Shapley), for emulating an OQ switch at a speedup of 2. The FLGS algorithm draws on previous work in [4]. A single, crucial, observation in this paper ensures that FLGS is fully local while those proposed earlier were believed not to be: we focus on emulating OQ switches which offer *port-level fairness* as opposed to other OQ scheduling algorithms such as FIFO, flow-level strict priority, or flow-level weighted fairness. Port-level policies operate at a coarser level of granularity than flow-level policies. The reason the port-level fairness assumption is crucial can be understood from Section IV.

It is now worth considering the FLGS algorithm from the practical stand-point. Port-level fair output scheduling policies are the most prevalent in practice (see [13], [14], [15], for example). These algorithms allow an operator to configure a router or a switch so that the bandwidth at an output can be shared according to weighted fair or strict priority policies by the various input ports. The general approach taken in designing schedulers for such systems is two-tiered: first, an output scheduler decides which input it wants the next packet from; then, the fabric scheduler obtains this packet from the input. Thus, there are two separate, but interlocking, schedulers: one for partitioning the output link bandwidth and the other for transferring packets from inputs to outputs.

[1]In fact, it is only at a speedup of 2 that iSLIP, when it is maximalized, is known to deliver 100% throughput. This follows from the fact that *any* maximal matching algorithm gives 100% throughput at speedup 2 [5].

0743-166X/07/$25.00 ©2007 IEEE

Speedup is used as the grease which, when adequately applied, smooths the interlocking mechanism. In this sense adequate speedup enables practical algorithms to "work;" without provable performance guarantees and without knowing exactly how much speedup is adequate. The only theoretical approach taken for integrating the two schedulers to date is the work on CIOQ switches, typified by Chuang et. al. [4]. But, as mentioned previously, these algorithms are too complicated for practice. The FLGS algorithm presented in this paper is the first *practical* integrated scheduler which provides provable performance guarantees at a speedup of 2.

Section V studies the performance of FLGS at speedups between 1 and 2 via simulations and compares its bandwidth partitioning ability to that of an OQ switch.

II. Model and notation

Consider an $N \times N$ crossbar switch such as the one shown in Figure 1. Assume that time is slotted and that cells arrive at the switch at the beginning of a time slot. We will assume that at most one cell arrives at an input port and at most one cell departs from an output port during a time slot. Cells arrive at the switch just at the beginning of a time slot and depart from the switch just prior to the end of a time slot.

Depending on the traffic pattern and the switching discipline, cells may have to wait at an input port before being switched to the correct output port, and may have to wait at an output port before departing. Each input has a buffer of infinite capacity for holding cells prior to switching them to their respective outputs. Likewise each output has an infinite capacity buffer for holding cells that will be placed on the outgoing line. The buffer at an input is partitioned into N "virtual output queues" (VOQs). The buffer at an output is partitioned into N "virtual input queues" (VIQs). The virtual output queue VOQ(i, x) holds cells arriving at input i destined for output x. The virtual input queue VIQ(i, x) holds cells which are waiting at the output port x and which arrived via input port i.

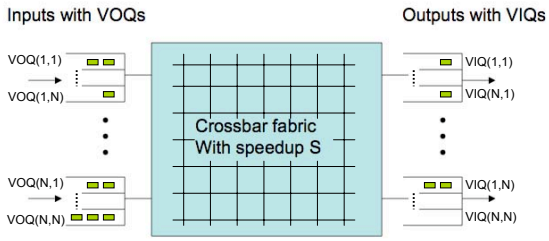

Fig. 1. *A CIOQ switch*

A "scheduling phase" consists of two parts: (a) the matching part, and (b) the switching part. During the matching part a matching algorithm, m, selects a matching between inputs and outputs in such a way that no input (respectively, output) may be matched to more than one output (respectively, input). During the switching part input i transfers a cell to output x if they are matched to each other and VOQ(i, x) is non-empty. The switch is said to have a speedup of s, where $s \in \{1, \ldots, N\}$, if during every time slot there are s scheduling phases.

Definitions and terminology: We will use $I(C)$ and $X(C)$ to denote the input and output ports of cell C, $a(C)$ and $d(C)$ to denote the its arrival and departure time respectively. A CIOQ switch *exactly emulates* an OQ switch if given the same arrival sequence, every cell C has the same departure time $d(C)$ from the two switches. A CIOQ switch is said to be consistent at time t if all the cells in a VIQ are older (i. e. arrived at the switch earlier) than all the cells in the corresponding VOQ.

III. Port-Ordered Policies

This section precisely defines a general class of policies, under the title "port-ordered policies." This class includes all those of practical interest, including arbitrary combinations of WFQ and strict priorities at the port level. Given the definition, we quite easily see that FIFO does not fall under this category.

A scheduling policy for an output queued switch is said to be *port-ordered* if

1) At any given time, every output port has a strict priority order defined on the input ports (and hence, on the VIQs). The priority order may be different at different output ports at any given time, and may also be different at the same output port at different times. The priority order may depend on the cells which have departed the output queue, but not on the cells which are currently in the switch.
2) During each departure phase, an output port transmits the oldest cell from the highest priority non-empty VIQ. We will use $r_{x,t}(i)$ to denote the rank of input port i in the priority list of output port x at time t. A rank of 1 denotes the highest priority.
3) The departure of a cell C from output port $X(C)$ at time t can only result in the port $I(C)$ moving down in the priority list of port $X(C)$. More formally, if i_1, i_2 are two input ports with $I(C) \neq i_1$ such that $r_{i_1,t} < r_{i_2,t}$, then $r_{i_1,t+1} < r_{i_2,t+1}$.

Port-ordered policies form a natural class, and include many algorithms of practical interest. The FIFO scheduling policy is not port-ordered, because the ranking of input ports at an output depends on the arrival time of cells currently in the switch. Hence, FIFO violates property 1 above.

Consider a CIOQ switch which has exactly emulated an OQ switch with a port-ordered policy till time t, and which is consistent at time t. Let $R(x, t)$ denote the set of cells which are either buffered at an output port x of this CIOQ switch at time t or have departed port x at some earlier time $t' < t$. Let $A(x, t)$ be the set of input ports whose buffer currently contains at least one cell destined for x. For $i \in A(x, t)$, let $S_i(x, t)$ denote the set of cells currently buffered at input port i which are destined for x. Observe that a fully local computation at output port x is one which can be performed using only knowledge of $R(x, t)$ and $A(x, t)$ (i.e. without any knowledge of the $S_i(x, t)$'s)[2].

Let $C_i(x, t)$ be the cell from $S_i(x, t)$ which has the earliest departure time $t_i(x, t)$ in the port-ordered policy

[2]With speedup of two, there will be two scheduling phases during a time slot, and hence the quantity $V(x, t)$, $A(x, t)$ etc. may not be uniquely defined. We get around this notational difficulty by allowing t to be half-integral when used as an argument to quantities A, S_i, V, t_i, R.

being emulated. Let $V(x,t)$ denote the sequence of size $|A(x,t)|$ obtained by sorting the set A in increasing order of $t_i(x,t)$. We will call this sequence the *departure-sorted input list* for port x at time t. Informally, the departure-sorted list represents the order in which cells currently residing at the heads of the different VOQs for x will be needed by the output port x. The departure-sorted list $V(x,t)$ can be computed using a simple fully local algorithm, described below.

Algorithm LOCAL-PORT-ORDER**:** /*Computes $V(x,t)$*/

1) Ignore all the cells in VIQ(i,x) for all $i \notin A(x,t)$.
2) Assume that VOQ(i,x) is of size 1 for all $i \in A(x,t)$.
3) Assume there are no future arrivals into the switch.
4) Simulate the port-ordered policy of the OQ switch on this restricted set of cells.
5) Arrange the input ports in $A(x,t)$ in the order in which the (single) cell in VOQ(i,x) leaves the output port in the above simulation.

Lemma 1: Consider a CIOQ switch which has exactly emulated an OQ switch with a port-ordered policy till time t. Then an output port x can compute the set $V(x,t)$ using the fully local algorithm LOCAL-PORT-ORDER.

This lemma establishes a concrete connection between port-ordered policies and fully local algorithms. The simplicity of the algorithm is striking. For the three common port-ordered policies mentioned earlier (WFQ, Round-Robin, and Static-Priorities), the algorithm is even simpler. For Round-Robin, the algorithm arranges the ports in $A(x,t)$ in decreasing order of the sizes of their VIQs at x; ties are broken depending on the position of the Round-Robin pointer. WFQ is similar. For Static Priorities, the set $A(x,t)$ is simply sorted in the static order. In each of these cases, determining $V(x,t)$ does not add any additional complexity to the switch.

IV. THE FLGS ALGORITHM

We will now give fully local algorithms for constructing the preference lists of inputs and outputs such that at a speed-up of 2, any port-ordered policy can be exactly emulated using a CIOQ switch. We will assume an arbitrary (but fixed) port-ordered policy. We will assume that there is an arrival phase at the beginning of a time slot, followed by two scheduling phases, followed by a departure phase at the end of the time slot.

a) The Scheduling Algorithm:: During each scheduling phase, the switch computes a stable marriage of the input ports to output ports, using preference lists computed by each port. This can be performed using the Gale-Shapley algorithm [6] which only communicates request, grant and accept information between the ports.

The key point to note is that the computation of the preference lists is fully local. Indeed, the computation of the preference lists at the outputs requires *only* the output link scheduler, which even an OQ switch needs. The computation of the input preference lists requires *nothing*; a very simple, fixed rule called GBVOQ (for Group By Virtual Output Queue) suffices. The details follow.

b) Output ports:: During a scheduling phase at time t, the output port x computes the departure-sorted list $V(x,t)$ using the algorithm LOCAL-PORT-ORDER. Then, x uses $V(x,t)$ as its preference list for the stable marriage algorithm. During a departure phase, output port x sends the highest priority cell currently residing at x.

c) Input ports:: Input port i maintains a sorted list of the non-empty VOQs at i. During an arrival phase, if a cell arrives into an empty VOQ, the VOQ is inserted at the front of this sorted list. If a cell arrives into a non-empty VOQ, the sorted list does not change. During a scheduling phase, if a VOQ becomes empty, it is deleted from the list. This sorted list is used as i's preference list during the stable marriage algorithm. This is the GBVOQ scheme. Observe that GBVOQ is analogous to LIFO at the port level[3].

We now state our main theorem:

Theorem 1: FLGS exactly emulates any given port-ordered policy at speedup 2.

Proof: We will adapt the proof of Chuang *et al.* [4] to port-ordered policies. Define the Input Thread of a cell C (denoted IT(C)) queued at a VOQ as the total number of cells ahead of C (including C) in the preference list of the input port. More precisely, the input thread of C is the number of cells ahead of C in its VOQ plus the number of cells in the VOQs ahead of C's VOQ in the priority list maintained at the input port. Define the Output Cushion of C, denoted OC(C), as the number of cells waiting at C's output port which have an earlier departure time according to the port ordered policy being emulated. Observe that in the input port's preference list during a stable marriage phase, output ports are arranged in increasing order of input threads of the head-of-line cell in the corresponding VOQ.

Also, in an output port's preference list during the stable marriage phase, input ports are arranged in increasing order of output cushions of the head-of-line cell in the corresponding VOQ. Thus, by definition of a stable marriage [6], during each scheduling phase, for a head-of-line cell in any VOQ, either its input thread goes down or its output cushion goes up. Define the slackness of a cell C (denoted $L(C)$) as OC$(C)-$ IT(C). Hence, during each scheduling phase, the slackness of a head-of-line cell in any VOQ must go up by at least 1. Since any cell in the input thread/output cushion of a head-of-line cell of any VOQ is also in the input thread/output cushion of all other cells in that VOQ, we conclude that during a scheduling phase, the slackness of any cell goes up by at least 1.

During an arrival phase, the input thread of an existing cell can go up by at most 1. During a departure phase, the output cushion of an existing cell can go down by at most 1. Hence, the slackness of a cell can not decrease during the entire cycle of arrival, scheduling, scheduling, and departure, as long as this cell resides at an input port.

Now consider a cell C that arrives into an empty VOQ. The input thread of this cell immediately after arrival is 1. Its

[3]This is a counter-intuitive scheme since it gives priority to newly populated VOQs. But Chuang et. al. [4] have used it to prove that it can be used for emulation of OQ switches which use monotone output scheduling policies at a speedup of 2.

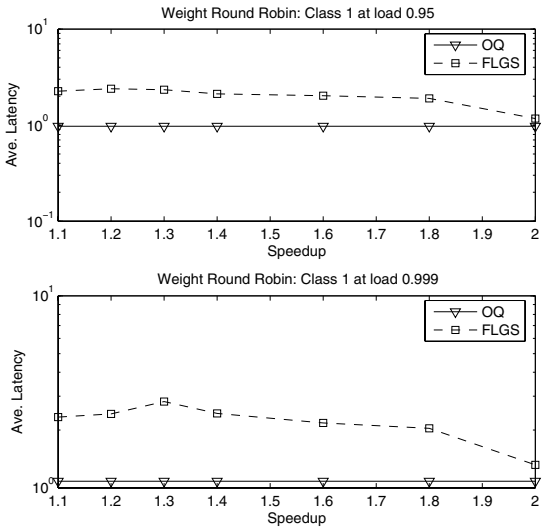

Fig. 2. *Weight Round Robin: Class 1 at loads 0.95 and .999*

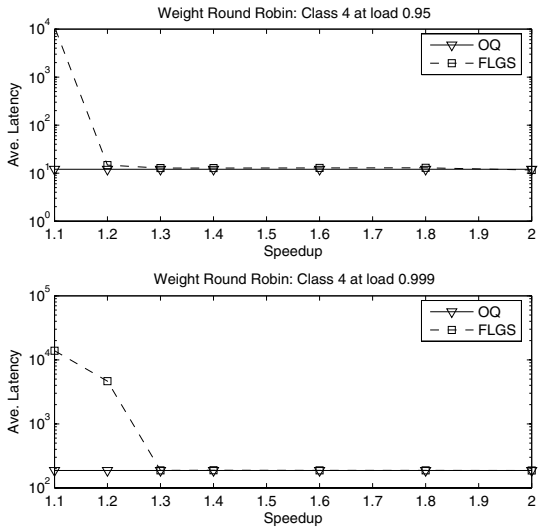

Fig. 3. *Weight Round Robin: Class 4 at loads 0.95 and 0.999*

output cushion is at least 0, and hence its slackness is at least -1. After the first scheduling phase, its slackness is at least 0. Hence, inductively, the slackness of this cell is at least 0 after the first scheduling phase as long as this cell resides at the input port. Now suppose the departure time of this cell is t. If this cell has not left the input port before or during the first scheduling phase at time t, its output cushion will be zero after the first scheduling phase. Therefore, this cell will have a slackness of at least 0, and hence an input thread of 0. Thus, the corresponding VOQ will be at the head of the preference list for both its input port and its output port. Therefore, any stable marriage must result in this cell being switched across during the second scheduling phase, and being available at the output port in time for departure.

Now consider a cell C that arrives into a non-empty VOQ. We will prove inductively that the head-of-line cell in C's VOQ always has a slackness of at least 0 after the first scheduling phase, and hence always makes the departure deadline. We have already established this for a cell which arrives into an empty VOQ, and hence the base case for our induction holds. Let C' be the cell just ahead of C. The output cushion of C is always at least as large as the output cushion of C' and the input thread of C is exactly one more than the input thread of C' as long as C' resides at the input side. When C' gets to the head of the VOQ, the slackness of C' is at least 0 (by the inductive hypothesis), and hence the slackness of C is at least -1. When C' gets switched to the output port, $\mathrm{IT}(C)$ goes down by 1 and $\mathrm{OC}(C)$ goes up by 1, increasing the slackness of C by two. This compensates for the slackness of C being -1 just before C' gets switched. Hence, from this time on, the slackness of C is at least 0 after the first scheduling phase, completing the inductive argument. ■

V. Simulation Results

In order to explore the performance of FLGS at speedups lower than 2, we performed a number of simulations to compare the bandwidth allocation of FLGS with an ideal output queued switch. A fixed-cell-size switch simulator with infinite VOQ and VIQ sizes was used to simulate a switch for 100K time steps. $\lambda_{i,j}$ represents the arrival rate into VOQ(i, j). We assume that the output port follows the same port-ordered policy for all schemes to determine which packets leave the switch. Of course the output port can only run this policy on the cells actually available for departure (i. e. those queued in a VIQ).

A. Weighted Round Robin

We measured the average latencies for cells flowing from different input ports to a specific output port for a Weighted Round Robin output scheduling policy, in order to compare the amount of bandwidth allocated to each input port. Figures 2–3 show measured average latencies for different classes of traffic for a 32x32 switch, supplied by a diagonal 4 traffic pattern where $\lambda_{i,i}$ = 0.1, $\lambda_{i,i+1}$ =0.2, $\lambda_{i,i+2}$ =0.3, and $\lambda_{i,i+3}$ = 0.35 (or 0.399 for a total load factor of 0.999).

The total arrival rate at an input or for an output equals 0.95 and 0.999 in the two separate experiments. The weight values at the output ports are adjusted inversely: $W_{i,i}$=4, $W_{i,i+1}$=3, $W_{i,i+2}$=2, $W_{i,i+3}$=1. The reason for this inverse ordering of weights is that a scheduler which does not integrate switching and bandwidth partitioning would give priority to the wrong class of packets during switching. Therefore, it would need a higher speedup to better track the OQ switch. The graphs show the latency of the traffic from input ports 1 and 4 to output 4. Class 1 traffic has the highest weight, equal to 4, and goes from input 4 to output 4. Class 4 traffic goes from input 1 to output 4 and has a weight of 1. As the figures show, FLGS can track the bandwidth allocation of the OQ switch extremely well for speedups more than 1.2.

B. Strict priorities

Figures 4–5, demonstrate a strict priority scheduling policy at the output port between four levels of traffic in a 10x10 switch. Class 1 is the highest priority while class 4 is the lowest. Traffic pattern is as follows: class 1, $\lambda_{i,i}$ =0.1; class 2: $\lambda_{i,i+1}$ =0.2, class 3: $\lambda_{i,i+2}$ =0.3 and class 4 $\lambda_{i,i+3}$ =0.35 (or 0.399 for a total load factor of 0.999).

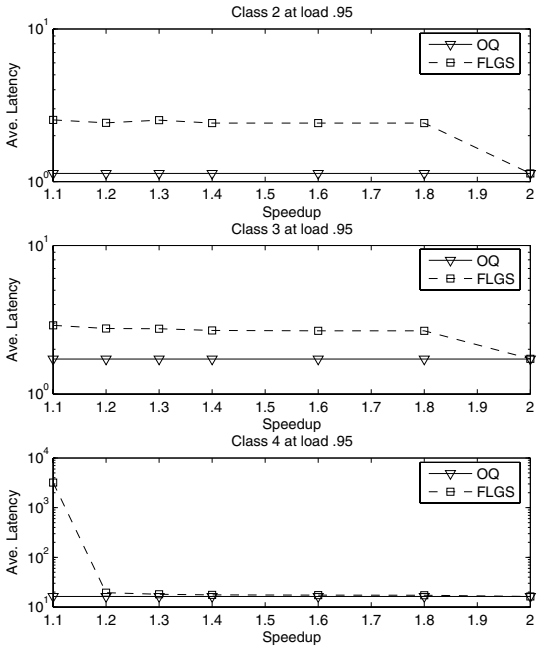

Fig. 4. *Strict Priority: Classes 2, 3, and 4 at load 0.95*

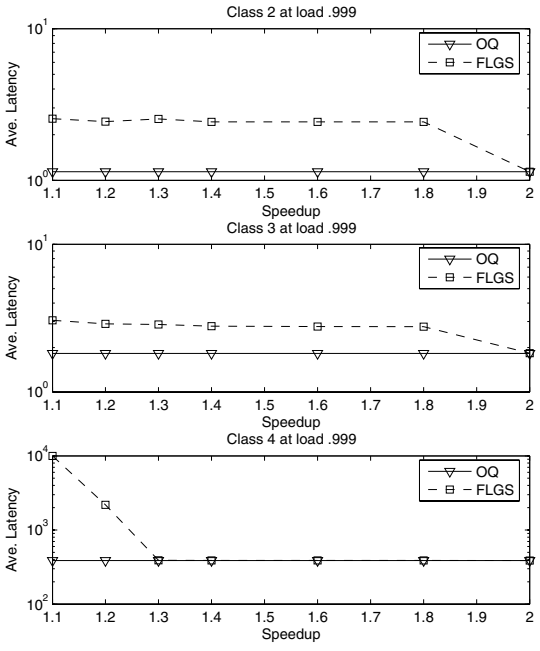

Fig. 5. *Strict Priority: Classes 2, 3, and 4 at load 0.999*

The graphs show the average latency experienced by traffic classes 2–4 in the OQ switch and FLGS for load factors of 0.95 and 0.999. The class numbers are as defined in previous section. Since FLGS integrates the scheduling of the switch fabric with the scheduling of the output port, it is able to redistribute the latency of the cells over the priority classes, such that classes with higher priority have lower latency at the cost of increased latency for lower priority classes.

VI. Conclusion

Motivated by the need for practical scheduling algorithms that integrate the switching and the output scheduling functions, we have introduced the class of fully local (FL), request-grant-accept (RGA) scheduling algorithms. We have noted that at speedup 1 no FL and RGA algorithm has been known to give 100% throughput, or that at speedup 2 no such algorithm has been known to emulate an OQ switch to date.

Given the practical importance of this class of algorithms, the main result of this paper — an FL and RGA algorithm, FLGS, for emulating an OQ switch — is interesting. The crucial observation of the paper is that port-level fair output scheduling policies (called port-ordered policies in the paper) are surprisingly amenable to an FL and RGA implementation. We prove that the FLGS algorithm emulates an OQ switch which uses any port-level fair policy at a speedup of 2. By simulations, we also find that the FLGS algorithm performs very competitively with respect to an OQ switch at speedups as low as 1.2.

References

[1] T. Anderson, S. Owicki, J Saxe and C. Thacker: "High speed switch scheduling for local area networks", *ACM Transactions on Computer Systems*, v 11, pp 319-352, Nov 1993.

[2] A. Charny: "Providing QoS Guarantees in Input Buffered Crossbar Switches with Speedup", PhD Thesis, MIT, 1998.

[3] A. Charny, P. Krishna, N. Patel and R. Simcoe: "Algorithms for Providing Bandwidth and Delay Guarantees in Input-buffered Crossbars with Speedup", *Presented at the 6^{th} IEEE/IFIP IWQ0S '98*, May 1998.

[4] S-T. Chuang, A. Goel, N. McKeown and B. Prabhakar: " Matching Output Queueing with a Combined Input Output Queued Switch", *IEEE Journal of Selected Areas in Communication*, 17:1030–1039. A short version appears in *Proc. Infocom '99*.

[5] Dai J., Prabhakar B., "The throughput of data switches with and without speedup", *IEEE INFOCOM 2000*, vol. 2, Tel-Aviv, Israel, Mar. 2000, pp. 556-564

[6] D. Gale, and L.S. Shapley, "College Admissions and the stability of marriage", *American Mathematical Monthly*, vol.69, pp.9-15, 1962.

[7] P. Giaccone, B. Prabhakar and D. Shah: "Randomized scheduling algorithms for input-queued switches," IEEE Journal of Selected Areas in Communication special issue on "High-performance electronic switches/routers for high-speed Internet," Vol. 21, No. 4, pp. 642-655, May 2003.

[8] P. Krishna, N. Patel, A. Charny and R. Simcoe: "On the Speedup Required for Work-conserving Crossbar Switches", *IEEE Journal of Selected Areas in Communication*, 17:1057-1066. *Presented at the 6^{th} IEEE/IFIP IWQ0S '98*, May 1998.

[9] N. McKeown: "iSLIP: A Scheduling Algorithm for Input-Queued Switches", *IEEE Transactions on Networking*, v.7, pp 188-201, April 1999.

[10] N. McKeown, V. Anantharam, J. Walrand: "Achieving 100% Throughput in an Input-Queued Switch", *INFOCOM '96*, pp.296-302.

[11] B. Prabhakar and N. McKeown: "On the speedup required for combined input- and output-queued switching", to appear in *Automatica*.

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

[13] Cisco Systems' MDS 9000 Switch Data Sheet: http://www.cisco.com/en/US/products/hw/ps4159/ps4358/products_data_sheet09186a00801ce945.html

[14] Cisco Systems' GSR 12000 Data Sheet: http://www.cisco.com/en/US/products/hw/routers/ps167/products_configuration_guide_chapter09186a0080513905.html#wp1051245

[15] Juniper Networks' M640 Router Data Sheet: http://www.jnx.com/products/mseries/100042.html