Tracking the Behavior of an Ideal Output Queued Switch Using an Input Queued Switch With Unity Speedup

Amir Gourgy, Ted H. Szymanski
Department of Electrical and Computer Engineering
McMaster University, Hamilton Ontario L8S 4K1, Canada
amir@grads.ece.mcmaster.ca, and teds@mail.ece.mcmaster.ca

Abstract— We address the problem of fair scheduling of packets in Internet routers with input-queued (IQ) switches. We present new performance metrics for IQ switches with unity speedup. Scheduling in IQ switches is formulated as *tracking* the behavior of an ideal output-queued (OQ) switch that provides optimal performance. We introduce several performance metrics that measure the difference between the ideal performance provided by an ideal OQ switch and an IQ switch with unity speedup. A key performance metric is the notion of "lag" between an IQ switch and an ideal OQ switch. Using the proposed metrics as design criteria, we present a suite of scheduling policies for IQ switches with unity speedup that provide better performance than existing scheduling policies in the literature, with comparable complexity.

I. INTRODUCTION

There is a tremendous demand for Internet core nodes to provide quality-of-service (QoS) guarantees for multimedia services, and to provide high switching capacity that makes use of the virtually unlimited bandwidth of optical fibers. The Internet's success depends on the deployment of high-speed switches and routers that meet these two demands.

On the one hand, the demand of QoS guarantees can be met using output-queued (OQ) switches, which can provide optimal throughput. In addition, much research effort, considering algorithms such as the weighted fair queueing (WFQ) family (e.g., [19]) has been devoted to packet scheduling at output ports to support fair bandwidth sharing that provides delay bounds for regulated traffic. However, OQ for an $N \times N$ switch requires the switching fabric and memory to run up to N times faster than the line rate; unfortunately, for large N or for high-speed data lines, memories with sufficient bandwidth are not available.

On the other hand, the fabric and the memory of an input-queued (IQ) switch need only to run as fast as the line rate. This makes input queueing very appealing for switches with fast line rates or with a large number of ports. But IQ switching can suffer from head-of-line (HOL) blocking, which limits the throughput to just 58.6%, if each input maintains a single FIFO [11]. One method that has been proposed to reduce HOL blocking is to increase the *speedup* of a switch. A switch with a speedup of S can remove up to S packets from each input and deliver up to S packets to each output within a *time slot*, where a time slot is the time between packet arrivals at an input port.

A theoretical result [5] established that an $N \times N$ combined input-and output-queued (CIOQ) switch with a speedup of two could exactly *emulate* an $N \times N$ OQ switch for any traffic pattern of input cells. Emulation occurs at every time instance if, under identical inputs both systems produce identical departures. Unfortunately, the complexity of the scheduling algorithm presented in [5] renders OQ switch emulation infeasible (see [14], [15] for a discussion of the complexity). The speedup requirement translates to a smaller time available for the execution of the arbitration algorithm. In a hardware implementation, reduction of the available time by a factor of two poses a substantial problem, although the difference does not seem significant asymptotically; it translates to a requirement of doubling the operating frequency of the arbiter, which might not be practically achievable. Furthermore, Minkenberg [18] have shown that exact emulation of an OQ using a CIOQ switch is possible only if the CIOQ have infinite output buffers.

Most commercial high-performance switches and routers (e.g., CISCO 1200[2], BBN [21], Lucent Cajun [3] family, or Avici TSR45000 [1]) use IQ switches. Most of these high-speed switches are built around a crossbar switch that is configured using a centralized scheduler designed to provide high throughput and use a fixed-length cell as a transfer unit. Fixed-length switching technology is widely accepted for achieving high switching efficiency such that variable-length packets are segmented into fixed-length cells at the inputs and are reassembled at outputs. We assume fixed-length cell scheduling for the remainder of this paper[1].

In this paper we consider scheduling policies in an IQ-crossbar switch with a unity speedup. Given that an IQ switch requires at least a speedup of two to exactly emulate an OQ switch [5], an IQ scheduling policy with a unity speedup can not exactly emulate the behavior of an OQ switch, under all possible traffic patterns. Consequently, we formulate scheduling in an IQ switch as the problem of *tracking* an ideal OQ switch. We introduce new performance metrics that measure the difference between the ideal performance provided by an OQ switch and an IQ switch. Using these metrics as design criteria, we present a suite of scheduling policies for IQ switches with unity speedup that provide better

[1] The words *packet* and *cell* are used interchangeably for the remainder of this paper.

0-7803-8375-3/04/$20.00 © 2004 IEEE

performance than existing scheduling policies in the literature, with comparable complexity. Although in this paper we describe the case of tracking an ideal OQ switch implementing only a FIFO scheduling policy, our results can be easily extended for other non-FIFO scheduling policies.

This paper is organized as follows. Section II formulates scheduling in an IQ switch with unity speedup as tracking the behavior of an ideal OQ switch. Section III describes the benefits of tracking the behavior of an ideal OQ switch. In section IV, we present a suite of scheduling policies for tracking the behavior of an ideal OQ switch with various performance and implementation complexities. The performance of these scheduling policies is evaluated through simulation for different traffic models and compared to several known scheduling policies in the literature in section V. Section VI provides our conclusions.

II. Problem Formulation

We consider an $N \times N$ ideal OQ switch that implements scheduling policy π_{OQ} and an IQ [2]switch with unity speedup that implements scheduling policy π_{IQ}. Let the average cell arrival rate at input i for output j be λ_{ij}. We assume that incoming traffic is admissible; that is, $\sum_{i=1}^{N} \lambda_{ij} < 1$, and $\sum_{j=1}^{N} \lambda_{ij} < 1$. The arrival process is identical to both switches. The goal is to find a scheduling policy π_{IQ} that *tracks* the behavior of the ideal OQ switch as close as possible, where we define what *tracking* means more precisely after introducing some definitions.

Given that an IQ switch requires at least a speedup of two to exactly emulate an OQ switch [5], an IQ scheduling policy with a unity speedup can not exactly emulate the behavior of an OQ switch, under all possible traffic patterns. In general, arriving cells to the IQ switch implementing π_{IQ} will depart at some later time than the ideal OQ switch implementing π_{OQ}. Consequently, we say that an IQ switch implementing π_{IQ} lags the behavior of the ideal OQ switch implementing π_{OQ}.

A. Definition of Terms

Here we make precise some of the terminology used throughout this paper.

Definition 1 Ideal departure time (IDT): The ideal departure time for a cell c [$IDT(c)$] is the time slot at which c will depart from an ideal OQ switch implementing π_{OQ}.

Definition 2 Actual departure time (ADT): The actual departure time (ADT) for a cell c [$ADT(c)$] is the time slot at which c departs from the switch under consideration (i.e., IQ implementing π_{IQ})

[2] When we refer to IQ switch with unity speedup, we include designs that employ combined input output queueing with unity speedup.

Definition 3 Cell Lag (CL): The cell lag for a cell c [$CL(c)$] is the difference between the ideal departure time and the actual departure time. Precisely,

$$CL(c) = \begin{cases} ADT(c) - IDT(c), ADT(c) > IDT(c) \\ 0, otherwise \end{cases}$$

We assume that every cell entering the switch will eventually depart at some time in the future; that is, $\forall c, ADT(c) < \infty$. To simplify the notation, let X denote the random variable given by $CL(c)$.

In addition, we define the cell lag for a cell c given the current time slot n [$CL(c,n)$] as the difference between the ideal departure time and the current time slot. Precisely,

$$CL(c,n) = \begin{cases} n - IDT(c), n > IDT(c) \\ 0, otherwise \end{cases}$$

To the best of our knowledge, we are the *first* to define this notion of lag between an IQ switch and an OQ switch.

Definition 4 Scheduling Policy Lag Mean (μ_{lag}): The lag mean for a scheduling policy π [$\mu_{lag}(\pi)$] is defined as the expectation or mean of the random variable X; that is, $\mu_{lag}(\pi)$ is the average lag of all cells departing from a switch implementing scheduling policy π. $\mu_{lag}(\pi) = E[X]$. The lag mean represents the additional delay provided by scheduling policy π_{IQ} than that provided by an ideal OQ switch.

Definition 5 Scheduling Policy Maximum Lag ($\max_{lag}$): The maximum lag for a scheduling policy π [$\max_{lag}(\pi)$] is defined as $\forall c, \max_{lag}(\pi) = \max(X)$

Definition 6 Scheduling Policy Lag Variance (σ^2_{lag}): The lag variance for scheduling policy π [$\sigma^2_{lag}(\pi)$] represents the variance in the lag between the switch implementing scheduling policy π and the ideal OQ switch; that is, $\sigma^2_{lag}(\pi) = E[(X - \mu_{lag})^2]$. Note that, $\sigma^2_{lag}(\pi)$ reflects the additional jitter provided by the scheduling policy π than that provided by an ideal OQ switch.

Although other metrics can be similarly defined, we consider the previous metrics sufficient to characterize the tracking criteria of an ideal OQ switch. The goal of a scheduling policy π_{IQ} can be characterized by any of the tracking metrics defined previously or a combination of them.

Remark: Although an IQ switch with unity speedup lags the behavior of an ideal OQ switch, for efficiency purposes a cell may occasionally depart from an IQ switch earlier than an ideal OQ switch; for example, consider a 2×2 switch at a specific time slot such that the two most lagging cells for its outputs (e.g., outputs 1, and 2) reside at the same input port (e.g., input 1). Because the scheduling policy can transfer at most one cell from each input port (e.g., input 1), another cell with a future ideal departure time might be selected from the other input port (e.g., input 2) to be transferred across the switch for efficiency purposes. However, in general an IQ

switch lags an OQ switch and π_{IQ} can be designed to be strictly lagging.

III. Motivation

In an ideal OQ switch arriving packets are immediately available at the outgoing link. Consequently, the only shared resource in an OQ switch is the outgoing link of the switch and packets contend for access to the outgoing link (output contention). In an IQ switch packets are queued at the input port of the switch and they must first contend for access to the switch fabric (input contention), before contending for the outgoing link; that is, in an IQ switch, there are two shared resources: the switch fabric and the outgoing link.

All present IQ scheduling policies resolve input and output contention using heuristics such as using a round-robin scheme at both the input and output to solve the contention fairly, or using the packet's age (i.e., time in the switch) to resolve contention. All these schemes can be seen as an approximation to the ideal case of an OQ switch, where all the outgoing links are independent and packets are served independently in each outgoing link; that is, by tracking the behavior of an ideal OQ switch and minimizing the lag, we automatically resolve input and output contention in a fair manner and eliminate any starvation problem of inputs that other scheduling policies have to carefully handle.

We emphasize that significant research effort (e.g., [7], [19], [20]) has been done in developing scheduling policies for ideal servers that provide bounded latency, jitter, and end-to-end delay for traffic flows. Unfortunately, the real Internet does not consist only of ideal servers, but rather of heterogeneous servers (i.e., non ideal IQ and CIOQ servers, and ideal OQ servers). By tracking the behavior of an ideal server, we approximate its behavior as close as possible and attempt to bound the performance difference between the ideal server and an IQ switch.

IV. Tracking Scheduling Policies

We consider the case of $\pi_{OQ} = FIFO$. The architecture of our IQ switch is shown in Figure 1.

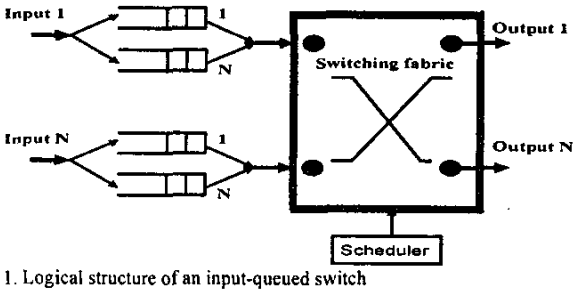

Fig. 1. Logical structure of an input-queued switch

We use virtual output queueing (VOQ) at each input port of the switch and a crossbar as the switching fabric. For $\pi_{OQ} = FIFO$, arriving cells at the IQ switch can be immediately assigned an IDT using a simple parallel prefix circuit [10] (i.e., a ranker); Let N_j be the number of cells in the ideal OQ switch destined to output j. The IQ switch uses N rankers such that each ranker calculates the number of cells present in the ideal OQ switch; specifically, each ranker j uses a variable N_j such that at the beginning of each time slot

$$N_j = \begin{cases} N_j - 1, N_j > 0 \\ 0, otherwise \end{cases}$$

Note that the subtraction of one in the previous equation accounts for one (cell/time slot) departure in the ideal OQ switch. For every new cell c arriving at time slot n destined to output j, ranker j assigns $IDT(c) = n + N_j$ and updates $N_j = N_j + 1$. For an $N \times N$ switch, we use the following notational conventions:

i an input, $1 \le i \le N$

j an output, $1 \le j \le N$

$VOQ(i, j)$ is the VOQ at input i and buffers cells destined for output j. $HOL(i, j)$ is the head-of-line cell at $VOQ(i, j)$.

First, we present a scheduling policy called *maximum weighted lag* (MWL) that is simple to describe, but has a high implementation cost. It serves as solid base for developing other practical scheduling policies that approximate its performance. Second, we present a scheduling policy that iteratively minimizes the maximum lag at a lower cost than MWL. Third, we present a maximal weighted lag scheduling policy that is readily implemented in hardware and provides excellent performance. Simulation results for all tracking scheduling policies are presented in section V.

A. Maximum Weighted Lag Scheduling Policy

Maximum weighted lag (MWL) is based on the implementation of a *maximum bipartite weight-matching algorithm* (MWM)[4]. A maximum weight matching on a bipartite graph with weighted edges is defined as a set of edges between input and output nodes with the maximum total weight among all possible sets satisfying the constraint that any input node is matched to at most one output node.

At every time slot n, we associate a weight $W(i, j)$ to every $VOQ(i, j)$ such that $W(i, j) = CL(HOL(i, j), n)$; that is, $W(i, j)$ is maximum lag of an HOL packet in $VOQ(i, j)$. The maximum weighted lag scheduling policy finds a matching M that maximizes $\sum_{(i,j) \in M} W(i, j)$ and can be found by solving an equivalent network flow problem [4]. The best implementation of a MWM has running time complexity $O(N^3 \log(N))$ on a sequential model [4]. We use a traditional implementation of a maximum bipartite graph matching. However, our contribution is in judiciously using the cell lag values as the edge weights. Previous work on MWM considered only the weight equaling to either some function of the occupancy of the VOQs (i.e., number of packets' in each VOQ) or the waiting time of the cell at the head of line of each VOQ (e.g., [13], [17], [22], [24], [25], and [26]). Consequently, these algorithms do not necessarily track the behavior of an ideal OQ switch and cells' departure time may

arbitrarily deviate from the ideal case if the arrival traffic is bursty. In addition, using the occupancy of the VOQs as the edge weight can lead to starvation of certain inputs [17].

Because MWL computes the matching with the maximum possible *total* weight during every time slot, it aims at minimizing the lag mean (μ_{lag}). This does not necessarily imply that the maximum lag is minimized. Although this algorithm is too complex to implement in practice, it serves as a reference model for which other approximation algorithms are developed.

B. Iterative Min Max-Lag Scheduling Policy

This scheduling policy iteratively minimizes the maximum lag (iMML) by performing a matching between an input port for the cell with maximum lag amongst all cells in the switch and its corresponding output port until no more matches can be performed.

iMML can be implemented using a request-accept paradigm using an arbiter at each input and output port such that in each iteration, each unmatched input sends a request to the output corresponding to its most lagging packet. Each output arbiter then examines its requests and sends a grant to the input arbiter with the most lagging packet. The input and output arbiter are considered matched. However, an output arbiter may break a match if it receives a more lagging request than its current matched input arbiter in the future. This "backtracking" procedure requires N^2 iterations in the worst case on a sequential model. The running time complexity of iMML is $O(N^2 \log N)$ on a sequential model. This scheduling policy is equivalent to stable-marriage matching [9], where the preference of each input port is the lag of its VOQs. (We name it *iterative min max-lag* because it is more descriptive). The best known parallel algorithm for the stable marriage problem is due to Feder et al. [8] and has a running time complexity $O(\sqrt{n}\log^3 n)$ and uses n^4 processors on a Concurrent Read Concurrent Write (CRCW) PRAM model.

MWL and iMML can be seen as tracking the behavior of an ideal OQ switch with different tracking criteria. The former tracking policy's objective is to minimize the lag mean and the latter's objective is to minimize the maximum lag. The two scheduling policies are related such that total weight of a stable marriage is at least half the total weight of a maximum-weighted matching [12]. Consequently, depending on the tracking criterion we aim to optimize, we can use either MWL or iMML.

C. Iterative Maximal Lag Scheduling Policy

Because iMML can break matches performed in previous iterations of the algorithm, it generally requires a large number of iterations to form a stable matching. In addition, this breaking of matchings made earlier in the matching process is also difficult to implement in practice. Consequently, we explore using a simpler greedy scheme based on *maximal matching*. A maximal matching is one that adds connections incrementally, without removing connections made earlier in the matching process. Iterative maximal lag (iML) is essentially a greedy version of iMML without backtracking. Initially all input and output arbiters are unmatched, then in each iteration:

1. **Request**: Each unmatched input sends a request to every unmatched output for which it has a queued cell.
2. **Grant**: If an unmatched output receives any requests, it chooses the request with the most lagging cell and sends a grant to this input.
3. **Accept**: If an unmatched input receives any grants, it chooses the grant for its most lagging cell and sends an accept signal to this output. The input and output arbiter are considered matched.

The algorithm executes until either no more matches can be made or a fixed number of iterations are performed. On a CRCW model, the algorithm requires $O(N)$ iterations such that each iteration is $O(N)$ computations. This algorithm is readily implemented in hardware.

V. Simulation Results

The performance of MWL, iMML, and iML are evaluated for a 16×16 switch under uniform and bursty traffic models. We compare the behavior of our scheduling policies the following scheduling policies: iSLIP [16], and iDRR [23]. Our choice of these scheduling policies is a balance between the extensively studied algorithms with excellent performance, commercially implemented (i.e., iSLIP) and recent published results in the literature (i.e., iDRR). These schemes are based on a hybrid of maximal matching and weighted round robin priority schemes, with complexity comparable to iML. Consequently, these scheduling policies take up to N iterations in the worst case on a CRCW model. For comparison purposes, all maximal matching based scheduling policies were executed until a maximal match was found.

The proposed tracking criteria (i.e., μ_{lag}, $\max_{lag}$, σ^2_{lag}) are evaluated in addition to the average cell delay for each traffic model.

A. Uniform Traffic Model

As shown in Fig. 2., all our proposed scheduling policies achieve 100 percent throughput under i.i.d. Bernoulli traffic.

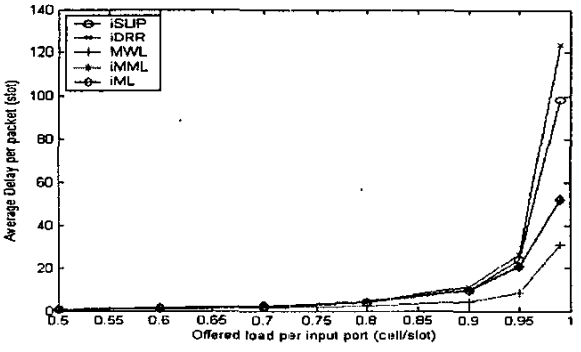

Fig. 2 Average Cell Delay of iSLIP, iDRR, MWL, iMML, and iML under uniform traffic

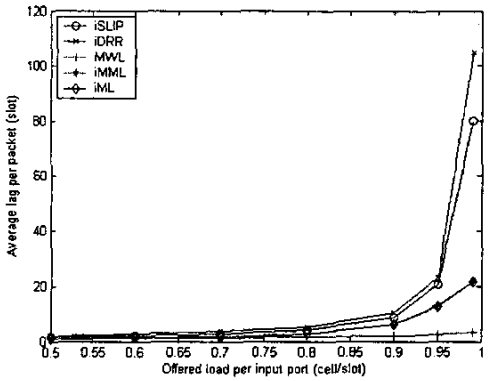

Fig. 3 Average Lag of iSLIP, iDRR, MWL, iMML, and iML under uniform traffic

In Fig. 3, MWL achieves the least lag mean compared to other scheduling policies (albeit at a significantly higher implementation cost), followed by iMML and iML with virtually identical performance. Finally, iSLIP and iDRR have the highest lag mean. The same trend occurs for the maximum lag as shown in Figure 4.

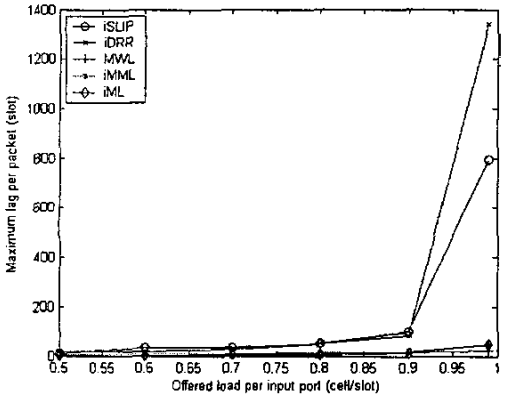

Fig. 4 Maximum Lag of iSLIP, iDRR, MWL, iMML, and iML under uniform traffic.

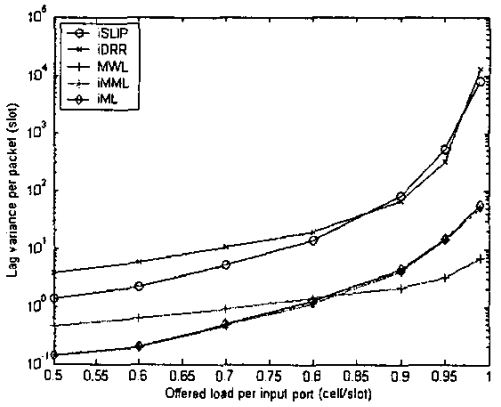

Fig. 5 Lag variance of iSLIP, iDRR, MWL, iMML, and iML under uniform traffic.

As shown in Fig. 5, the lag variance of the proposed scheduling policies is almost an order of magnitude better than both iSLIP and iDRR.

B. Bursty Traffic Model

Real network traffic is self-similar and highly correlated such that cells tend to arrive in bursts [6]. The performance of the tracking scheduling policies was evaluated under a bursty traffic model such that each input port is connected to a burst source that generates traffic–cells are generated using a 2-state Markov process that alternates between busy and idle states. The process remains in the busy and idle states for a geometrically distributed number of cell times. When the server is in the busy state, cells arrive at the beginning of every cell time and all with the same set of destinations. This traffic model was also used in [16]. A burst size of 16 was used.

Although iMML, iML provide comparable delay to iSLIP, and iDRR as shown in Fig. 6, iMML and iML provide smaller lag mean, lag variance, and maximum lag as shown in Figures 7, 8, 9, respectively. Note that MWL always provides the best performance at high traffic loads, albeit at a higher implementation cost.

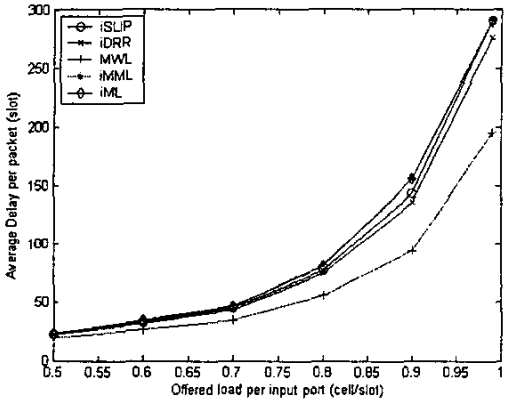

Fig. 6 Average Cell Delay of iSLIP, iDRR, MWL, iMML, and iML under bursty
traffic

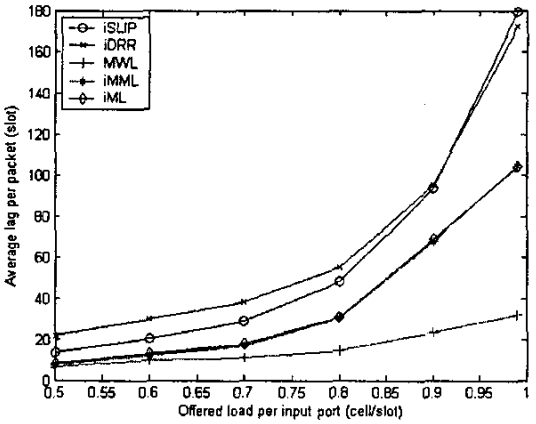

Fig. 7 Average Lag of iSLIP, iDRR, MWL, iMML, and iML under bursty traffic

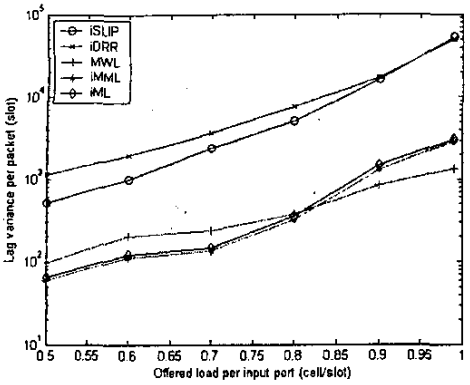

Fig. 8 Lag variance of iSLIP, iDRR, MWL, iMML, and iML under bursty traffic.

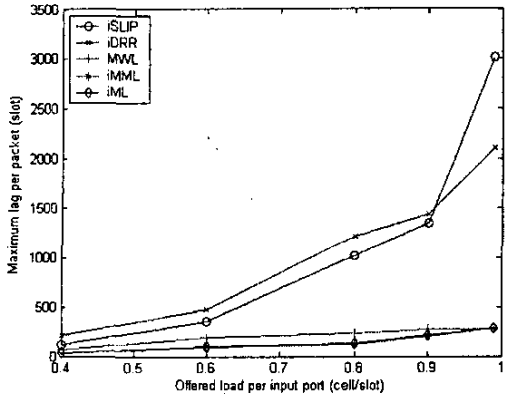

Fig. 9 Maximum Lag of iSLIP, iDRR, MWL, iMML, and iML under bursty traffic.

VI. CONCLUSION

IQ switches are commercially used in most Internet routers due to their capability of operating at high line speeds with lower memory bandwidth requirement than OQ switches. In this paper, we addressed the issue of fair scheduling in Internet routers with IQ switches. We formulated switch scheduling in an IQ switch with unity speedup as tracking the behavior of an ideal OQ switch. By tracking the behavior of an ideal OQ switch, an IQ switch resolves input and output contention fairly, eliminates any starvation of inputs, and approximates the ideal behavior of an OQ switch as close as possible. We introduced several metrics that quantify the difference between the ideal behavior of an OQ and an IQ switch. Using those metrics as design criteria we proposed a suite of scheduling policies with varying design criteria and implementation complexities. Finally, we showed through simulation that our proposed scheduling policies provide superior performance compared to the best proposed scheduling policies in the literature.

REFERENCES

[1] [1] Avici Systems, Inc., Billercia, MA. [Online]. Available: http://www.avici.com

[2] Cisco 12000 Series-Internet Routers [Online]. Available: http://www.cisco.com

[3] Lucent Technologies, Inc., Holmdel, NJ. [Online]. Available: http://www.lucent.com/

[4] Ravindra K. Ahuja, Thomas L. Magnanti , James B. Orlin. Network Flows: Theory, Algorithms, and Applications. Prentice Hall; 1 edition (February 18, 1993)

[5] S-T. Chuang et al., " Matching Output Queueing with a Combined Input Output Queued Switch", IEEE J. Select. Areas Commun., vol. 17, no. 6, pp 1030- 1039.

[6] Mark E. Crovella and Azer Bestavros, ``Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes", IEEE/ACM Transactions on Networking, 5(6):835-846, December 1997.

[7] R. L. Cruz, "A calculus for network delay, Part II: Network analysis," IEEE Trans. Inform. Theory, vol. 37, pp. 132-141, 1991.

[8] T. Feder, N. Megiddo, and S. Plotkin. "A sublinear parallel algorithm for stable matching." Fifth ACM-SIAM Symposium on Discrete Algorithms, p. 632-637 (1994).

[9] D. Gale and L. S. Shapley, "College admissions and the stability of *marriage*," *Amer. Math. Monthly*, vol. 69, pp. 9-15, 1962.

[10] C.P. Kruskal, R. Rudolph, and M. Snir. The Power of Parallel Prefix. IEEE Trans. on Comp., C-34(10), October 1985.

[11] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queueing on a space-division switch," IEEE Trans. Commun., vol. 35, pp. 1347-1356, Dec. 1987.

[12] A. Kam, K.-Y. Siu, R. A. Barry, and E. Swanson, "A cell switching WDM broadcast LAN with bandwidth guarantee and fair access," J. Lightwave Technol., vol. 16, pp. 2265-2280, Dec. 1998.

[13] I. Keslassy and N. McKeown, "Analysis of Scheduling Algorithms That Provide 100% Throughput in Input-Queued Switches," Proc. of the 39th Annual Allerton Conference on Communication, Control, and Computing, Monticello, Illinois, October 2001.

[14] P. Krishna, N. Patel, A. Charny, and R. Simcoe, "On the speedup required for work-conserving crossbar switches," IEEE J. Select. Areas Commun., vol. 17, pp. 1057-1066, June 1999.

[15] B. Magill, C. Rohrs, R. Stevenson, "Output-Queued Switch Emulation by Fabrics With Limited Memory", in IEEE Journal on Selected Areas in Communications, pp.606-615, May. 2003.

[16] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," IEEE/ACM Trans. Networking, vol. 7, no. 2, pp. 188 -201, April 1999.

[17] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch", IEEE Trans. Commun., vol. 47 no. 8, pp. 1260 -1267, Aug. 1999.

[18] C. Minkenberg, "Work-conservingness of CIOQ packet switches with limited output buffers," IEEE Commun. Letters, vol. 6, pp. 452 -454, Oct. 2002.

[19] A. Parekh and R.G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case," IEEE/ACM Trans. Networking, vol. 1, no. 3, pp. 344-357, June 1993.

[20] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the multiple node case," ACM /IEEE Trans. Networking , vol. 2, pp. 137-150, April 1994.

[21] C. Partridge et al., "A 50-Gb/s IP router," IEEE/ACM Trans. Networking, vol. 6, no. 3, pp. 237-248, June 1998.

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

[23] Xiao Zhang, and Laxmi N. Bhuyan, " Deficit Round Robin Scheduling for Input-Queued Switches," IEEE J. Selected Areas in Comm., vol. 21, no. 4 , pp. 584 -594, May 2003.

[24] Leonardi E., Mellia M., Neri F., Ajmone Marsan M., "Bounds on Average Delays and Queue Size Averages and Variances in Input-Queued Cell-Based Switches", IEEE INFOCOM 2001, Alaska, April 2001, pp.1095-1103.

[25] Leonardi E., et al., "On the stability of Input-Queued Switches with Speed-Up ", ACM /IEEE Trans. Networking , vol. 9, pp. 104-118, Feb. 2001.W.-K. Chen, *Linear Networks and Systems* (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135.

[26] E. Leonardi et al., "Bounds on delays and queue lengths in input-queued cell switches", JACM, vol. 50, pp. 520-550, July 2003.