An Efficient Randomized Algorithm for Input-Queued Switch Scheduling

Devavrat Shah, Paolo Giaccone, Balaji Prabhakar
devavrat@cs.stanford.edu, giaccone@polito.it, balaji@isl.stanford.edu
Dept. of CS, Stanford University; Dept. of EE, Politecnico di Torino; Depts. of EE and CS, Stanford University

Abstract— **The essential problem in the design of high-performance schedulers for input-queued switches is the determination, in each time slot, of a good matching between inputs and outputs for the transfer of packets. The rapid increase of line rates is making this very difficult: finding good matchings takes time, and there is very little time at the highest line speeds. A similar difficulty arises when designing schedulers for switches with a large number of ports.**

Randomized algorithms have proved particularly effective in providing good scalable solutions to problems where decisions need to be made within a limited amount of time and/or with little information. The main idea of randomized algorithms is simply stated: Basing decisions on a few randomly chosen samples is often a good surrogate for basing decisions with complete knowledge of the state.

This paper is about the design of randomized algorithms for switch scheduling. We begin by examining the difficulty of implementing well-known deterministic solutions like the maximum weight matching algorithm, discuss a simple randomized proposed by Tassiulas [1], develop a suite of novel randomized algorithms, and discuss their theory and performance.

I. Introduction

Many networking problems suffer from the so-called 'curse of dimensionality'; that is, although excellent (even optimal) solutions exist for these problems, they do not scale well to high speeds and/or to large systems. In a variety of other situations where the scalability of deterministic algorithms is poor, randomized versions of the same algorithms are easier to implement and provide surprisingly good performance. For example, compelling demonstrations are provided in the recent papers of [2], [3], [4] for load balancing, and [5] for document replacement in web caches. Other examples and a good introduction to the theory of randomized algorithms may be found in the book by Motwani and Raghavan [6].

This paper focuses on the application of randomization to the design of input-queued (IQ) switch[1] schedulers.

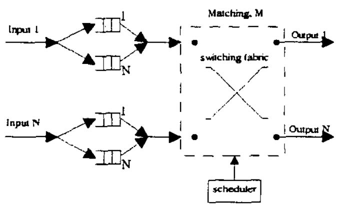

Fig. 1. Logical structure of an input-queued cell switch

Figure 1 shows the logical structure of an $N \times N$ IQ packet switch. We assume the switch operates on fixed-size cells (or packets). Each input has N FIFO 'virtual output queues' (VOQs), one for each output. This VOQ architecture avoids performance degradation due to the head-of-the-line blocking phenomenon [8].

In each time slot, at most one cell arrives at each input and at most one cell can be transferred to an output. When a cell with destination j arrives at input i, it is stored in the virtual output queue, denoted Q_{ij}. Let the average cell arrival rate at input i for output j be λ_{ij}. The incoming traffic is called *admissible* if $\sum_{i=1}^{N} \lambda_{ij} < 1, \forall j$ and $\sum_{j=1}^{N} \lambda_{ij} < 1, \forall i$. In words, these conditions ensure that no input or output is oversubscribed.

The scheduling problem can be modeled as a matching problem in a bipartite graph, with N input nodes and N output nodes. The edge from input i and output j is present if Q_{ij} is non-empty and is given the weight w_{ij} which equals the length of Q_{ij}. Given the transfer constraints in the switching fabric, a matching for this bipartite graph is a valid schedule. For example, Figure 2 shows a weighted bipartite graph and one valid matching (or schedule). Note that a valid matching can be seen as a permutation of the N outputs, and in this paper we will use the words *schedule, matching* and *permutation* interchangeably.

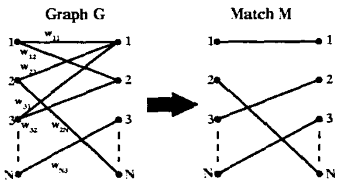

Fig. 2. Bipartite graph description of the scheduling problem

It is known that the maximum weight matching (MWM) algorithm delivers a throughput of up to 100% [8], [9], [10], and provides low delays by keeping queue-sizes small. However, it is too complex to implement since it requires $O(N^3)$ iterations [11] in the worst-case. Therefore, an efficient design of the overall system (scheduler and switching fabric) requires the best possible compromise between ease of implementation and goodness of throughput and delay performance.

As pointed out in [12], the specific issues in high-performance router design depend on whether the router operates in backbone networks or in enterprise networks. Routers in backbone networks, which interconnect a small number of enterprise networks, have few ports operating at a high line rate. Hence a good scheduling algorithm in this scenario needs to have a low time-complexity. The routers used in enterprise networks typically have a large number of ports connected to slower speed lines.

[1]We take for granted the goodness of the IQ architecture for very high speed and for large-sized switches. Several references (e.g. [8], [13]) attribute this to the minimal memory bandwidth requirement of the IQ architecture in comparison to the output-queued and shared-memory architectures.

0-7695-1357-3/01 $10.00 © 2001 IEEE

Although smaller line rates allow more time for scheduling, this time is consumed by a greater number iterations stemming from the large number of ports.

Several good switch scheduling algorithms have been proposed; notably iSLIP [13], iLQF [14], RPA [15] and MUCS [16]. With centralized implementations the run-time of these algorithms is $O(N^2)$ or more [17]; but by adopting parallelism and pipelining (that means adding spatial-complexity in hardware) their time-complexity can be decreased considerably. However, the performance of these algorithms is poor compared to MWM under non-uniform input traffic: they induce very large delays and their throughput can be less than 100%. Further, it is unlikely that solutions which intrinsically possess an $O(N^2)$ run-time complexity can be scaled for implementation in high-speed and large-sized switches.

This paper attempts to design low-complexity switch schedulers by exploiting the power of randomized algorithms. We begin by observing some features of the switch scheduling problem that can be exploited, discuss earlier randomized approaches, and obtain, through a series of 'evolutionary steps', some new low-complexity randomized switch scheduling algorithms.

A. The main features of our approach

Our approach is based on the following observations:
(a) The state of the switch, captured, for example, by its queue-lengths, does not change by much between two consecutive time slots. Thus, it is likely that good matchings at times t and $t+1$ are quite closely related in that the heavier edges in the one are likely to be in the other. This suggests it is possible to use the matching at time t for devising the matching at time $t+1$, and there is no need for computing matchings from scratch in each time slot.
(b) A randomly generated matching can be used to improve the matching used at time t for obtaining the matching at time $t+1$.
(c) Most of the weight of a matching is typically contained in a small number of edges. Thus, it is more important to choose edges at random than it is to choose matchings at random. Equally, it is more important to remember the few good edges of the matching at time t for use in time $t+1$ than it is to remember the entire matching at time t.

The recent paper of Tassiulas [1] proposes a very simple randomized algorithm that is mainly based on features (a) and (b). The algorithm can be described as follows. At time $t+1$ choose a matching R uniformly at random from the $N!$ possible matchings. Compare the weight of R with the matching M used at time t, use the heavier matching as the schedule at time $t+1$, and remember it for the next time slot. He proved that this algorithm achieves up to 100% throughput. However, as we will see later, the delays experienced by packets under this matching can be very large. Essentially, it is feature (c) that needs to be exploited for controlling delays.

We use all three features (a), (b) and (c) to devise an efficient randomized algorithm, called LAURA. It can be proved that it achieves a throughput of up to 100%. Simulations show that it provides delays close to that of MWM, and outperforms all other known low-complexity scheduling algorithms. LAURA needs an external source of randomness to obtain random matchings each time. This can cause some difficulty in implementation. To overcome this we propose an enhanced version of LAURA, called SERENA, which exploits the randomness present in the arrivals process to determine good random matchings. Fortuitously, this also improves the performance, since the arrivals are precisely what increase the weight of edges. Using them leads to better (heavier) schedules.

II. Discussion of Randomized Approaches

Using simulations we present a series of steps for determining the right criteria for designing efficient randomized schemes. We begin with some naive schemes, progressively make design decisions for improving their performance, and end up with the schemes LAURA and SERENA. We now describe the simulation setup that we shall use.

A. The Simulation Setting

The Switch: The size of the switch, N, equals 32. Each VOQ has a maximum capacity, $Q_{\max}$, of 10000. Buffers are not shared. Excess packets are dropped.
Input Traffic: Packets arrive at inputs according to independent and identically distributed (i.i.d.) Bernoulli processes. All inputs have equal normalized load, and the corresponding load factor is denoted by ρ. In the following we abbreviate $k \mod N$ to $|k|$.

The loading matrices considered are:
1. Uniform: $\lambda_{ij} = \rho/N \ \forall i, j$. This is the most commonly used test traffic in the literature.
2. Diagonal: $\lambda_{ii} = 2\rho/3N$, $\lambda_{i|i+1|} = \rho/3N \ \forall i$, and $\lambda_{ij} = 0$ for all other i and j. This is a very skewed loading, in the sense input i only has packets outputs i and $|i+1|$. It is more difficult to schedule this type of traffic than it is to schedule uniform loading, since arrivals favor the use of only two matchings out of the 32! possible matchings.
3. Logdiagonal: $\lambda_{ij} = 2\lambda_{i|j+1|}$ and $\sum_i \lambda_{ij} = \rho$. For example, the distribution of the load at input 1 across outputs is: $\lambda_{1j} = 2^{N-j}\rho/(2^N - 1)$. This type of load is more balanced than diagonal loading, but clearly more skewed than uniform loading. Hence, the performance of a specific algorithm will worsen as we change the loading from uniform to logdiagonal to diagonal.
Performance measures: Algorithms are compared on the basis of the mean input queue-lengths they induce, delays can be computed using Little's formula.

We let the simulation run until the confidence interval of the estimated average delay reaches a relative width of 1% with probability ≥ 0.95. The estimation of the confidence interval uses the *batch means* approach.

B. Random I

In this and the next few sections we present various randomized algorithms. Due to limitations of space, we shall consider their performance only under *diagonal* loading. This type of loading is particularly discriminating with randomized algorithms, because it requires them to find good matchings *at random* from a large space of possible matchings.

We proceed with the first randomized algorithm, called Random I. It is the most obvious randomized algorithm and works as follows:

(a) Every time pick a matching R uniformly at random from all possible $N!$ matchings.
(b) Use R as schedule.

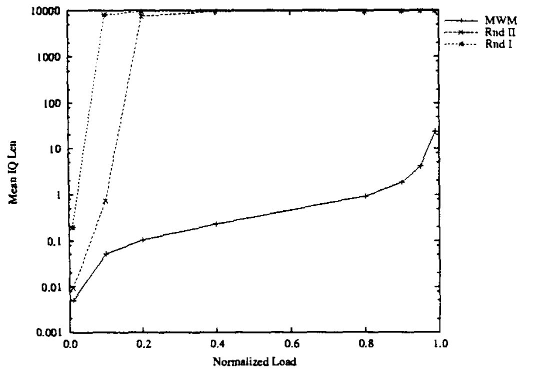

Fig. 3. *Random I has a very poor throughput and delay performance. Random II is better than Random I, but still quite bad when compared with MWM. This figure was generated under diagonal traffic pattern.*

The performance of this algorithm, displayed in Figure 3, shows that the average queue-length under *diagonal traffic* pattern is excessive when the load $\rho > .06$.

C. Random II

An obvious refinement of the previous algorithm, which we call Random II, is the following:
(a) Choose $d > 1$ matchings uniformly at random in each time slot.
(b) Use the highest of these matchings as the schedule.

For a choice of $d = 32$, Figure 3 shows that Random II performs better than Random I (as expected). However, its performance is still quite poor compared to MWM.

D. Random III

This algorithm, originally proposed by Tassiulas [1], works as follows:
(a) Let $S(t)$ be the schedule used at time t.
(b) At time $t + 1$ choose a matching $R(t + 1)$ uniformly at random from the set of all $N!$ possible matchings.
(c) Let the schedule at time $t + 1$, $S(t + 1)$, be the heavier of $S(t)$ and $R(t + 1)$.
As mentioned earlier, observe that Random III exploits the fact that state of the input buffers don't change by much during successive time slots. Tassiulas [1] shows that this makes Random III a stable matching; that is, it delivers a throughput of up to 100%. It clearly outperforms Random II (see Figure 4) in terms of delay. But, when compared with MWM, the delays it induces are still very large even when the load is approximately 40% (again, see Figure 4).

E. Random IV

We now use the observation that most of the weight of matching is typically carried in a few edges, and therefore it is better to remember edges between iterations than it is to remember entire matchings. To elaborate on this point, note that under uniform loading most edges have similar weights and it does not matter which matching is used. This is also the main reason that most

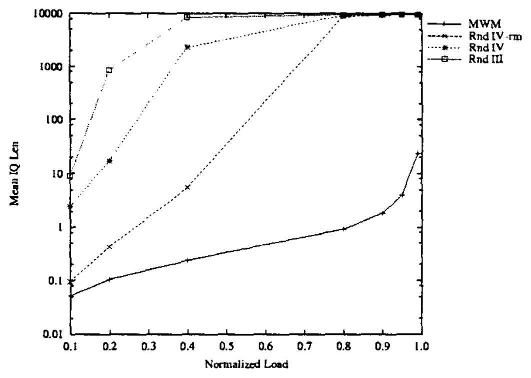

Fig. 4. *Random III, although theoretically stable, results in very high delays. Random IV outperforms Random III since it keeps the most significant edges from one time slot to the other. This figure was generated under diagonal traffic pattern.*

algorithms perform well under uniform loading. But, when the loading is non-uniform, edge-weights are highly skewed: most of the weight of a randomly chosen matching is carried in very few edges. Algorithms which exploit this, therefore, typically outperform algorithms which don't.

Definition 1. Let $\mathcal{F}_\eta(M)$ be the minimal set of edges in a matching M carrying at least η fraction of the total weight of M. Let $|\mathcal{F}_\eta(M)|$ denote the cardinality of $\mathcal{F}_\eta(M)$. Here $0 < \eta \leq 1$, is the *selection factor*.

As the next step in our evolutionary development, consider the algorithm Random IV described below:
(a) Let $S(t)$ be the matching used at time t.
(b) Compute $\mathcal{F}_\eta(S(t))$.
(c) At time $t + 1$, let $R(t + 1)$ be the matching which first uses the edges in $\mathcal{F}_\eta(S(t))$. This leaves $N - |\mathcal{F}_\eta(S(t))|$ input/output nodes unmatched. $R(t+1)$ connects these unmatched input/output nodes using a randomly chosen matching.
(d) Let $S(t + 1)$ equal the heavier of $R(t + 1)$ and $S(t)$.

Random IV can be generalized to Random IV-*rm*, that stores m matchings from the past and considers r random matchings, obtained by applying the phase (c) of Random IV r times independently, to improve each of these m matchings.

Figure 4 shows the performance improvement given by Random IV and Random IV-rm with $\eta = 0.5$ and $m = r = N$. The idea of keeping the "best" edges of a matching, from one time slot to another, is promising and we will use it in our innovative scheduler called LAURA.

III. LAURA

LAURA is mainly based on the following ideas:
1. Use "good" schedules from previous time, and avoid computation from scratch every time.
2. Obtain good random matching using a very different technique, which is sensitive to higher weight edges.
3. Instead of choosing better of two different schedules, *merge* them to obtain better solution.

The complete algorithm is described below. But we would like to note that, in the interest of space, we do not describe some details of the algorithm. Let $M_1, \ldots, M_S$ be S distinct matchings remembered from past. Let $\psi(M)$ denote weight of matching M at the current time. Every time do the following:

(i) Obtain V random matchings $X_1, \ldots, X_V$ from V independent trials of procedure RANDOM, which is described in next section.

(ii) Obtain higher weight schedules, $M'_{ij} = \text{MERGE}(M_i, X_j)$, for $1 \leq i \leq S, 1 \leq j \leq V$. Procedure MERGE is described in next section.

(iii) Let $\tilde{M}_i = \arg\max_j\{\psi(M'_{ij})\}$.

(iv) Let $M_{max} = \arg\max_i\{\tilde{M}_i\}$, which is used as schedule. In case of Max-LAURA version, the *maximized* version of M_{max} is used.

(v) Retain only the S matchings with the highest weight among all $S \times V$ schedules M'_{ij}.

Next we explain the two procedures used by LAURA.

A. RANDOM *Procedure*

The random selection procedure finds a random matching dependent on the weight matrix. At the same time, the random selection cannot be based on a non-uniform random selection based on the weights, since it is too complex to be implemented. To obtain an effective and simple random selection procedure, RANDOM runs in multiple stages to obtain a weight dependent schedule, while at each stage it uses random matching generated independent of weights. RANDOM procedure is described as follows: Initially, all inputs and outputs are marked as *unmatched.* The following steps are repeated in each of I iterations:

(i) Let $1 \leq i \leq I$ be the current iteration number. Let $k \leq N$ be the number of *unmatched* input-output pairs. A random matching $X_i(k)$ of this *unmatched*,bipartite graph is chosen uniformly at random from the $k!$ possibilities.

(ii) If $i < I$, retain the edges corresponding to $\mathcal{F}_\eta(X_i(k))$ and mark the nodes they cover as *matched.* If $i = I$, then retain all edges of $X_i(k)$.

The above procedure gives a complete matching with N edges.

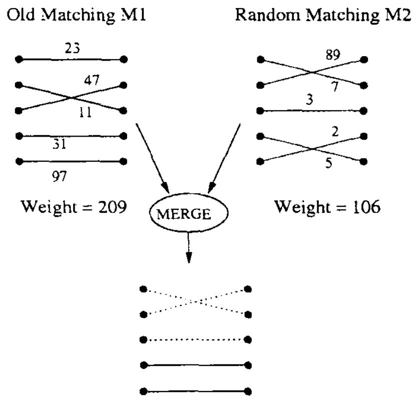

Fig. 5. Merging example for matching M1 and M2. The weight of the final matching is always greater or equal to the maximum among M1 and M2.

B. MERGE *Procedure*

MERGE runs on two matchings M_1 and M_2 and returns a matching $\tilde{M}$. It is described as follows.

Let $G' = M_1 \cup M_2$, in other words G' is a bipartite graph with the edges obtained by the union of matchings M_1 and M_2. Let $\tilde{M}$ be initially a bipartite graph with no edges.

Phase A. Mark all N input and output nodes of G' as *unmarked.* Repeat the next six steps till all nodes in G' are *marked*; after visiting at most $2N$ edges, the phase ends:

(a) Let v be an *unmarked* input node. Set path $P_v = \emptyset$. Let $\psi(P_v)$ denote weight of path P_v. Initially, set $\psi(P_v) = 0$.

(b) Let $(v, w) \in M_1$. Add (v, w) to P_v, and set $\psi(P_v) = \psi(P_v) + \psi(v, w)$.

(c) Let $(w, u) \in M_2$. Add (w, u) to P_v, and set $\psi(P_v) = \psi(P_v) - \psi(w, u)$.

(d) If $u = v$ stop. Else, repeat (b)-(c) for with u in place of v, and update P_v accordingly.

(e) Let $M_1(P_v)$ denote the edges of M_1 that belong to P_v, and similarly denote $M_2(P_v)$. If $\psi(P_v) \geq 0$, set $\tilde{M} = \tilde{M} \cup M_1(P_v)$; else $\tilde{M} = \tilde{M} \cup M_2(P_v)$.

(f) If any node q is *unmarked,* start from (a) with q in place of v.

Phase B. Output $\tilde{M}$ as the solution, which has property $\psi(\tilde{M}) \geq \psi(M_1), \psi(M_2)$.

Figure 5 shows an example of merging of the two matchings $M1$ and $M2$.

C. *Stability Properties of LAURA*

We state the following theorem:

Theorem 1. *LAURA is a stable algorithm, i.e. it achieves 100% of throughput under any admissible traffic patterns.*

We omit the proof due to lack of space.

D. *Running Time of LAURA*

The worst case running time of LAURA is bounded by $O(VIN \log_2 N + SVN)$. In our proposed implementation, we set: $I = \log_2 N$. S and V are constant. In particular, in our implementation we set $S = 2$ and $V = 1$. Hence, the running time of the algorithm is: $O(N \log^2 N)$. This is quite low compared to the running time of $O(N^3)$ for MWM [11], of $O(N^{2.5})$ for Maximum Size matching [19] and of all other approximations proposed in literature [17].

Max-LAURA does extra work to make the matching maximal. If l of N nodes are left unmatched, then a simple algorithm to get maximal matching takes worst case time of $O(l^2)$. If l is small, this is negligible compared to the rest. From simulation study we find that, in most of the cases, $l \ll N$, which suggests that the additional work done by Max-LAURA is negligible.

E. *Robustness of LAURA*

We explored the sensibility of LAURA to several parameters, in order to understand its robustness when its complexity is decreased. We studied the sensibility to I, S, d_{MIN} and also we studied a derandomized version of it. In all the cases, we experienced always delays comparable to the original LAURA version.

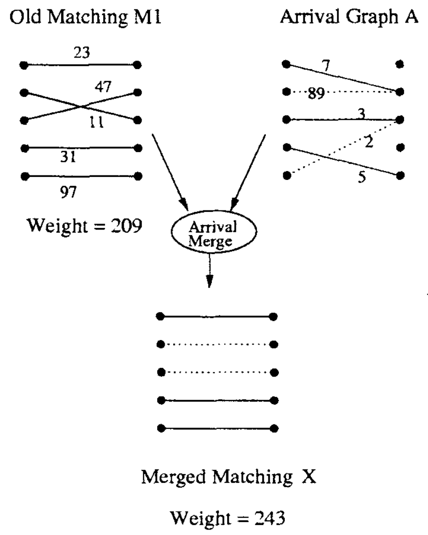

Fig. 6. ARRIVAL-MERGE example for matching $M1$ and arrival graph A. The final weight is always greater or equal to the weight of $M1$.

IV. SERENA

We now discuss a variant of LAURA, which uses the arrivals as source of randomness and an innovative merging algorithm. The randomization in LAURA is used to obtain unknown *heavy* edges with low complexity. Now observe that an edge becomes *heavy*, if its corresponding queue receives many arrivals and few services. Hence, the randomness provided by arrivals can be captured and exploited to find heavy edges. Whereas the basic version of LAURA merges the past schedule with a randomly generated matching, SERENA considers the edges that received arrivals in the previous time, and *merges* them with the past matching to obtain a higher weight matching. Note that merging of existing schedule and the arrival edges is not as simple as the MERGE procedure of LAURA since arrivals need not occur in a complete matching form. For this reason we denote it as ARRIVAL-MERGE procedure. Note that this procedure is very easy to be implemented since the randomness is obtained by observing only N edges. We do not discuss how the merging is done for this special case due to lack of space, but due to input constraints of one arrival per input, it can be done in $O(N)$ time.

SERENA which is briefly described here:

(a) Let $M(t)$ be matching used at time $t-1$.
(b) Let $A(t) = (A_{ij}(t))$ denote the arrival graph, where $A_{ij}(t) = 1$ indicates arrival, and $A_{ij}(t) = 0$, otherwise.
(c) Let $S(t) = \text{ARRIVAL-MERGE}(M(t), A(t))$, where ARRIVAL-MERGE is a special procedure, which we describe by an example in Figure 6.
(d) Use $S(t)$ as schedule, and $M(t+1) = S(t)$.

SERENA takes $O(VN + SVN)$, and for particular choice of V, S, it is $O(N)$.

We would like to note that SERENA is a *self-randomized* algorithm. It does not use any *external* randomization.

V. PERFORMANCE STUDY

Figures 7, 8 and 9 compare the performance of LAURA and Max-LAURA (using the settings of Table I) with some well known algorithms known in literature: iSLIP [13] (with N iterations) and iLQF [14] (with N iterations). LAURA shows poor delays for low load, because it is not maximal. By making it maximal, the Max-LAURA has delays as good as MWM even for low loads. LAURA and Max-LAURA outperform all the other approximating algorithms for high load under non-uniform traffic patterns.

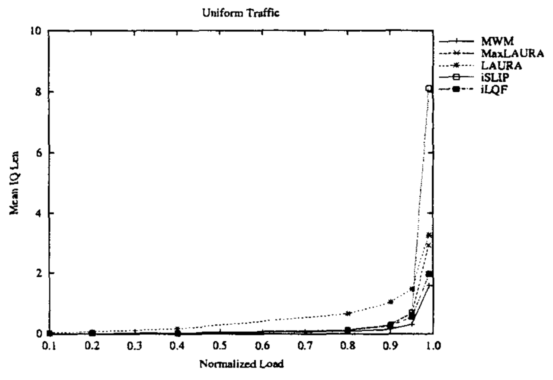

Fig. 7. For uniform traffic, all the considered algorithms are well behaved.

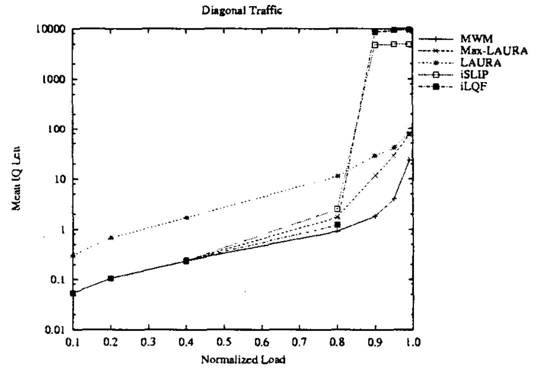

Fig. 8. For diagonal traffic, LAURA and Max-LAURA are able to reach the same throughput as MWM; Max-LAURA improves the performance of LAURA since it is maximal.

Parameter	Symbol	Value
random matching probes	V	1
stored matchings	S	2
iterations	I	5
selection factor	η	0.5

TABLE I
SIMULATION SETTINGS FOR LAURA AND MAX-LAURA

The Figure 10 compares the performances of SERENA and LAURA-S1 (i.e., only one stored matching). It shows that ARRIVAL-MERGE of SERENA outperforms the usual MERGE of LAURA which uses RANDOM procedure. But ARRIVAL-MERGE requires some what more complex data structure. The choice of SERENA and LAURA should be decided based on the

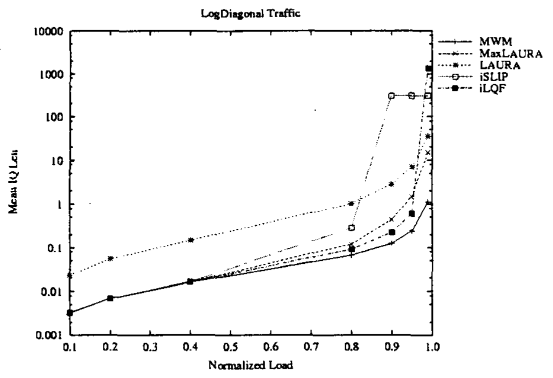

Fig. 9. For logdiagonal traffic, LAURA and Max-LAURA outperforms the other approximating algorithms for high load.

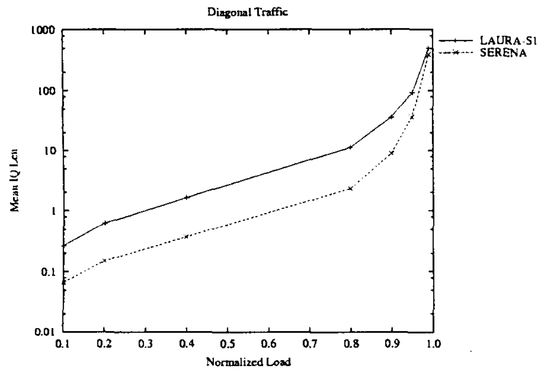

Fig. 10. SERENA is compared for diagonal traffic with LAURA using one stored matching ($S = 1$). In this case, the ARRIVAL-MERGING used in SERENA outperforms the MERGING of LAURA-S1.

overall system performance-design tradeoff.

VI. Conclusions

The paper presents a new randomized switch scheduling algorithm, called LAURA, that approximates the Maximum Weight Matching (MWM) algorithm. LAURA gives a throughput of 100% for all admissible Bernoulli i.i.d. inputs. Simulation studies show that it approximates the delays of MWM very well, and outperforms all known heuristics. Its run-time complexity is $O(N \log^2 N)$. We also presented another algorithm, called SERENA, which uses the randomness present in packet arrivals and hence does not need an external source of randomness. The run-time of SERENA is $O(N)$. Thus, as algorithms that provide good delay properties, SERENA (and LAURA) scale well with the switch size and provide a feasible approach for designing schedulers for high capacity routers.

References

[1] Tassiulas L., "Linear complexity algorithms for maximum throughput in radio networks and input queued switches", *IEEE INFOCOM'98*, vol. 2, New York, 1998, pp. 533-539

[2] Azar Y., Broder A., Karlin A.,Upfal E., "Balanced Allocations", *ACM STOC*, 1994, pp. 593-602,

[3] Mitzenmacher M., "The power of two choices in randomized load balancing", *PhD thesis*, University of California, Berkeley, 1996.

[4] Vvedenskaya N., Dobrushin R., Karpelevich F., "Queueing system with selection of the shortest of two queues : An asymptotic appro *Problems of Information Transmission*, 1996.

[5] Psounis K., Prabhakar B., "A randomized web-cache replacement scheme", *IEEE INFOCOM'01*, Anchorage, Alaska, April 22-26, 2001.

[6] Motwani R., Raghavan P., "Randomized algorithms", *Cambridge Univ. Press*, 1995

[7] Karol M., Hluchyj M., Morgan S., "Input versus output queuing on a space division switch", *IEEE Trans. on Communications*, vol. 35, n. 12, Dec. 1987, pp. 1347-1356

[8] McKeown N., Anantharan V., Walrand J., "Achieving 100% throughput in an input-queued switch" *IEEE Infocom '96*, vol. 1,San Francisco, Mar. 1996, pp. 296-302

[9] Tassiulas L., Ephremides. A., "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks". *IEEE Trans. on Automatic Control*, vol. 37, n. 12, Dec. 1992, pp. 1936-1948.

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

[11] Tarjan R.E., *Data structures and network algorithms*, Society for Industrial and Applied Mathematics, Pennsylvania, Nov. 1983

[12] Keshav S., Sharma R., "Issues and trends in router design", *IEEE Communications Magazine*, vol. 36, n. 5, May 1998, pp.144-151

[13] McKeown N., "iSLIP: a scheduling algorithm for input-queued switches", *IEEE Trans. on Networking*, vol. 7, n. 2, Apr. 1999, pp. 188-201

[14] McKeown N., "Scheduling algorithms for input-queued cell switches", *Ph.D. Thesis*, Un. of California at Berkeley, 1995

[15] Ajmone Marsan M., Bianco A., Leonardi E., Milia L., "RPA: a flexible scheduling algorithm for input buffered switches", *IEEE Trans. on Communications*, vol. 47, n. 12, Dec. 1999, pp. 1921-33

[16] Duan H., Lockwood J.W., Kang S.M., Will J.D., "A high performance OC12/OC48 queue design prototype for input buffered ATM switches", *IEEE INFOCOM'97*, vol. 1, Kobe, 1997, pp. 20-28.

[17] Ajmone Marsan M., Bianco A., Filippi E., Giaccone P., Leonardi E., Neri F., "On the behavior of input queuing switch architectures", *European Trans. on Telecommunications*, vol. 10, n. 2, Mar. 1999, pp. 111-124

[18] Goudreau M.W., Kolliopoulos S.G., Rao S.B., "Scheduling algorithms for input-queued switches: randomized techniques and experimental evaluation", *IEEE INFOCOM 2000*, vol. 3, Tel-Aviv, Mar. 2000, pp. 1634-1643

[19] Hopcroft J.E., Karp R.M., "An $n^{2.5}$ algorithm for maximum matching in bipartite graphs", *Society for Industrial and Applied Mathematics J. Comput.*, vol. 2, 1973, pp. 225-231