Efficient Randomized Algorithms for Input-Queued Switch Scheduling

High-performance schedulers for input-queued (IQ) switches must find, for each time slot, a good matching between inputs and outputs to transfer packets. At high line rates or for large switches, finding good matchings is complicated. A suite of randomized algorithms for switch scheduling provides performance comparable to that of well-known, effective matching algorithms, yet is simple to implement.

Devavrat Shah
Stanford University

Paolo Giaccone
Politecnico di Torino

Balaji Prabhakar
Stanford University

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 or large systems. In various other situations where deterministic algorithms' scalability is poor, randomized versions of the same algorithms are easier to implement and provide surprisingly good performance. For example, recent work in load balancing[1,2] and for documenting replacement in Web caches[3] provides compelling demonstrations of the effectiveness of these randomized algorithms. Motwani and Raghavan provide other examples and a good introduction to the theory of randomized algorithms.[4]

Here, we focus on applying randomization to the design of input-queued (IQ) switch schedulers. We take for granted the effectiveness of the IQ architecture for very high-speed and for large-sized switches. Several references attribute this effectiveness to the IQ architecture's minimal memory bandwidth requirement compared with output-queued and shared-memory architectures.

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 first-in first-out virtual output queues (VOQs), one for each output. This VOQ architecture avoids performance degradation from the head-of-the-line blocking phenomenon.[5]

In each time slot, at most one cell arrives at each input and at most one cell can transfer to an output. When a cell with destination output j arrives at input i, the switch stores it in the VOQ, denoted Q_{ij}. Let the average cell arrival rate at input i for output j be λ_{ij}. Incoming traffic is admissible if $\Sigma_{i=1}^{N} \lambda_{ij} < 1, \forall j$; and $\Sigma_{j=1}^{N} \lambda_{ij} < 1, \forall i$. In other words, these conditions ensure that no input or output is oversubscribed.

We can model the scheduling problem as a matching problem in a bipartite graph with

0272-1732/02/$17.00 © 2002 IEEE

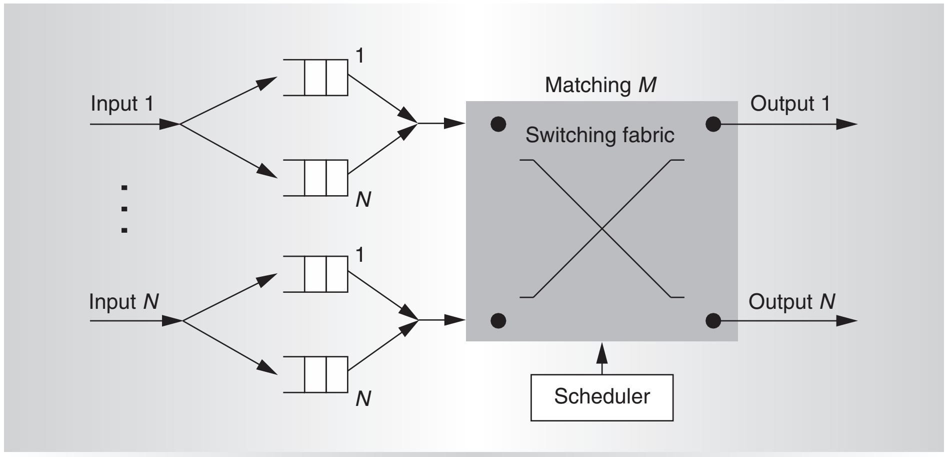

Figure 1. Logical structure of an input-queued cell switch.

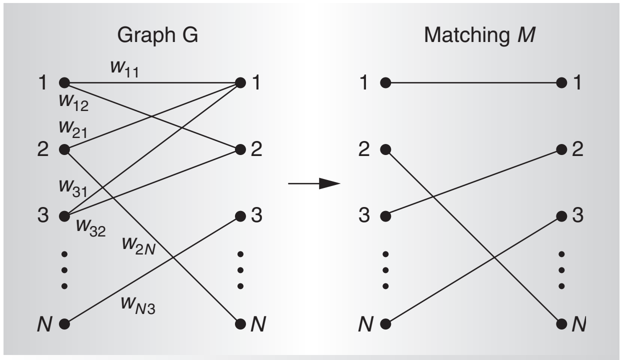

Figure 2. Bipartite graph of the scheduling problem.

N input nodes and N output nodes. The edge between input i and output j is present if Q_{ij} is nonempty; we give it 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). We can consider a valid matching as a permutation of the N outputs, and in this article, we use the words schedule, matching, and permutation interchangeably.

The maximum-weight matching (MWM) algorithm delivers a throughput of up to 100 percent[5,6] and provides low delays by keeping queue sizes small. However, it is too complex to implement because it requires $O(N^3)$ iterations 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 Keshav and Sharma pointed out, the specific issues in high-performance router design depend on whether the router operates in backbone or enterprise networks. Routers in backbone networks, which interconnect a few enterprise networks, have few ports operating at a high line rate. Hence, a good scheduling algorithm for this scenario must have a low time complexity.

Routers in enterprise networks typically have several ports connected to slower lines. Although slower line rates allow more time for scheduling, a greater number of iterations—stemming from the large number of ports—consumes this extra time.

Researchers have proposed several good switch scheduling algorithms, such as iterative SLIP (iSLIP),[7] iterative longest queue first (iLQF),[8] reservation with preemption and acknowledgment (RPA),[9] and matrix unit cell scheduler (MUCS)[10] With centralized implementations, the runtime of these algorithms is $O(N^2)$ or more. But by adopting parallelism and pipelining (which means adding spatial complexity in hardware) these algorithms can considerably reduce their time complexity.

However, under nonuniform input traffic the performance of these algorithms is poor compared to MWM: They induce long delays, and their throughput can be less than 100 percent. Furthermore, solutions that intrinsically possess an $O(N^2)$ runtime complexity are unlikely to scale for implementation in high-speed and large-sized switches.

Here, we discuss how to design low-complexity switch schedulers by exploiting the power of randomized algorithms.

Main features of our approach

We base our approach on the following observations:

- The state of the switch as captured by its queue lengths, for example, 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 weighted edges in one matching are likely to be in the other. This suggests that it is possible to use the matching at time t to devise the matching at time $t + 1$, eliminating the need to compute matchings from scratch in each time slot.
- We can use a randomly generated matching to improve the matching used at time t and use it for obtaining the matching at time $t + 1$.
- Most of a matching's weight is typically contained in a few 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.

Tassiulas recently proposed a very simple randomized algorithm, based mainly on the first two observations.[11] We can describe this algorithm as follows. At time $t + 1$, choose matching R uniformly and randomly from the $N!$ possible matchings. Compare R's weight with matching M used at time t, and use the more heavily weighted matching as the schedule at time $t + 1$. Remember this matching for the next time slot.

Tassiulas proved that this algorithm achieves up to 100 percent throughput. However, as we will see later, packets using this matching can experience long delays. Essentially, an algorithm must exploit our third observation to control delays.

We use all three observations to devise an efficient randomized algorithm, which we call Laura. We can prove that it achieves a throughput of up to 100 percent. 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, which can cause some implementation difficulties. To overcome this problem, 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 performance, because the arrivals are precisely what increase the weight of edges. Using them leads to better (more heavily weighted) schedules.

Discussion of randomized approaches

Using simulations, we present a series of steps for determining the correct 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 Laura and Serena schemes.

Simulation setting

We first must define a particular switch, input traffic, and performance measures.

Switch. Switch size N equals 32. Each VOQ has maximum capacity $Q_{\max}$ of 10,000 packets. The switch does not share buffers, and it also drops excess packets.

Input traffic. Packets arrive at inputs according to independent and identically distributed Bernoulli processes. All inputs have equal normalized load, and ρ denotes the corresponding load factor. In the following, we abbreviate k mod N as $|k|$. We considered three types of loading matrices:

- *Uniform.* In a uniform matrix, $\lambda_{ij} = \rho/N$ $\forall$ i,j. This is the most commonly used test traffic profile in the literature.
- *Diagonal.* A diagonal loading has $\lambda_{ii} = 2\rho/3$, $\lambda_{i|i+1|} = \rho/3$ $\forall$ i, and $\lambda_{ij} = 0$ for all other i and j. This loading is skewed in the sense that input i has packets only for outputs i and $|i + 1|$. This type of traffic is more difficult to schedule than uniform loading, because arrivals favor the use of only two matchings out of the 32! possible matchings.
- *Log diagonal.* For a log-diagonal loading, $\lambda_{ij} = 2\lambda_{i|j+1|}$ and Σ_i $\lambda_{ij} = \rho$. For example,

the load distribution 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, a specific algorithm's performance will become worse as the loading changes from uniform to log diagonal to diagonal.

Performance measures. We compared algorithms on the basis of the mean input queue lengths they induce and computed delays using Little's formula. The simulations ran until the confidence interval of the estimated average delay reached a relative width of 1 percent with probability ≥ 0.95. The estimation of the confidence interval uses the batch means approach.

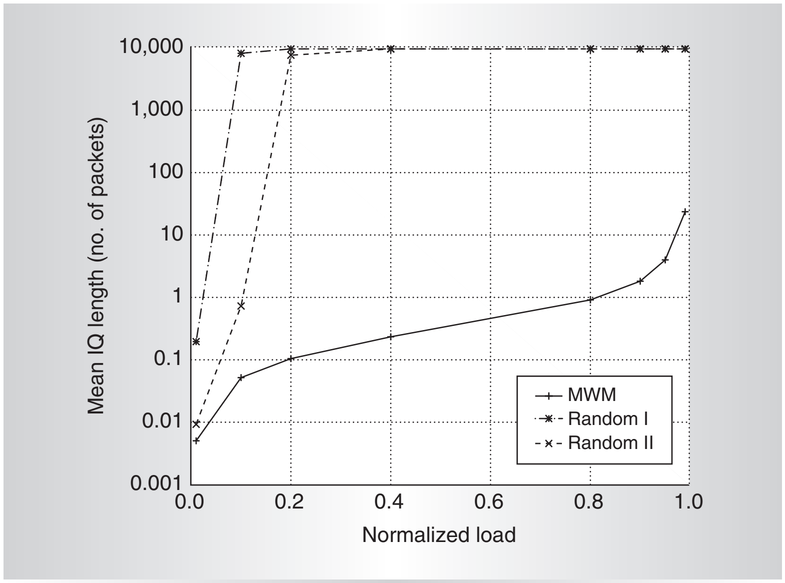

Figure 3. Random I has a very poor throughput and delay performance. Random II is better than Random I, but still performs quite poorly when compared with MWM. We generated this figure under a diagonal traffic pattern.

Random I

In this and the next few sections, we present various randomized algorithms. Due to space limitations, 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 randomly from a large space of possible matchings.

The first randomized algorithm, Random I, is the most obvious randomized algorithm and works as follows:

- For every time, pick matching R uniformly and randomly from all possible $N!$ matchings.
- Use R as the schedule.

For this algorithm, Figure 3 shows that the average queue length under a diagonal traffic pattern is excessive when normalized load $\rho > 0.06$.

Random II

An obvious refinement of the previous algorithm, which we call Random II, is the following:

- Choose $d > 1$ matchings uniformly and randomly in each time slot.
- Use the highest weighted of these matchings as the schedule.

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

Random III

This algorithm, originally proposed by Tassiulas,[11] works as follows:

- Let $S(t)$ be the schedule used at time t.
- At time $t + 1$, choose a matching $R(t + 1)$ uniformly and randomly from the set of all $N!$ possible matchings.
- Let the schedule at time $t + 1$, $S(t + 1)$, be the heavier weighted of $S(t)$ and $R(t + 1)$.

As mentioned earlier, Random III exploits the fact that the input buffers' states don't change by much during successive time slots. Tassiulas shows that this fact makes Random III a stable matching; that is, Random III delivers a throughput of up to 100 percent. This algorithm clearly outperforms Random II in terms of delay, as Figure 4 (next page) shows. But, when compared with MWM, the delays it induces are still very large even when the load is approximately 40 percent of maximum possible throughput.

Random IV

We now use the observation that just a few edges carry most of the matching's weight, and

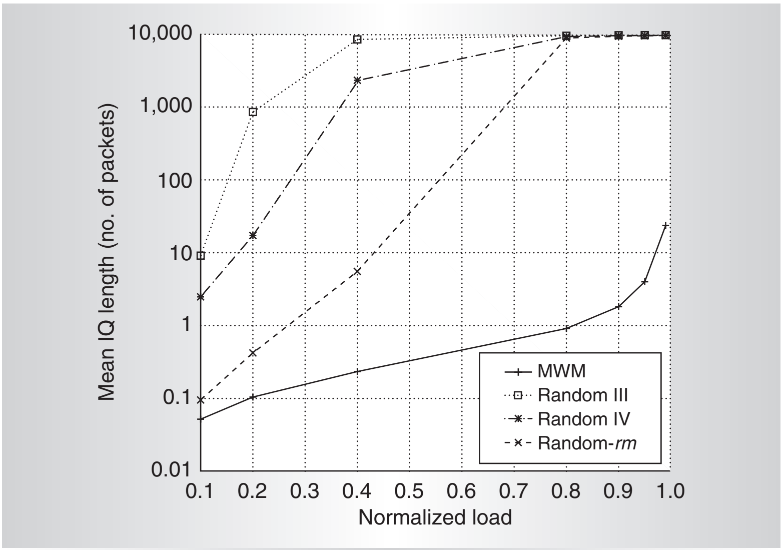

Figure 4. Diagonal traffic results. Random III, although theoretically stable, results in high delays. Random IV outperforms Random III because it keeps the most significant edges from one time slot to the other.

therefore it is better to remember edges between iterations than it is to remember entire matchings. Under uniform loading, most edges have similar weights, and it does not matter which matching we use. This is also the main reason that most algorithms perform well under uniform loading. But when the loading is nonuniform, edge weights are highly skewed: Just a few edges carry most of the weight of a randomly chosen matching. Algorithms that exploit this fact, therefore, typically outperform algorithms that don't.

Let $F_\eta(M)$ be the minimal set of edges in matching M carrying at least an η fraction of the total weight of M. Let $|F_\eta(M)|$ denote the cardinality of $F_\eta(M)$. Here $0 < \eta \leq 1$, where η is the selection factor.

As the next step in our evolutionary development, consider the following algorithm, Random IV:

- Let $S(t)$ be the matching used at time t.
- Compute $F_\eta[S(t)]$.
- At time $t + 1$, let $R(t + 1)$ be the matching that first uses the edges in $F_\eta[S(t)]$. This leaves $N - |F_\eta[S(t)]|$ input/output nodes unmatched. $R(t + 1)$ connects these unmatched input/output nodes using a randomly chosen matching.
- Let $S(t + 1)$ equal the heavier weighted of $R(t + 1)$ and $S(t)$.

We can generalize Random IV to Random IV-*rm* to improve each of the m matchings. Random IV-*rm* stores m matchings from the past and considers r random matchings, obtained by applying the third phase of Random IV r times independently.

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. We use it in our innovative scheduler, Laura.

Laura

We base Laura mainly on the following ideas:

- Use good schedules from a previous time, and avoid computation from scratch every time.
- Obtain good random matchings using a very different technique that is sensitive to higher-weight edges.
- Instead of choosing the better of two different schedules, merge them to obtain a better solution.

We next describe the complete algorithm, but, in the interest of space, we do not describe some details.

Let M_1, ..., M_S be S distinct matchings remembered from past time slots. Let $\psi(M)$ denote the weight of matching M at the current time. For every time slot, do the following:

- Obtain V random matchings X_1, ..., X_V from V independent trials of procedure **`Random`**, which we describe in the next section.
- Obtain higher-weight schedules, M'_{ij} = Merge(M_i, X_j), for $1 \leq i \leq S$, $1 \leq j \leq V$. We describe procedure **`Merge`** in a following section.
- Let $\tilde{M}_i = \arg\max_j[\psi(M'_{ij})]$.
- Let $M_{\max} = \arg\max_i(\tilde{M}_i)$, which we use as schedule. In the Max-Laura version, we use the maximized version of $M_{\max}$.
- Retain only the S matchings with the highest weight among all $S \times V$ schedules M'_{ij}.

Random procedure

This random selection procedure finds a random matching that depends on the weight matrix. At the same time, the random selection cannot be a nonuniform random selection based on the weights, because such a scheme is too complex to implement. 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 matchings generated independently of weights. We describe the **`Random`** procedure as follows. Initially, we mark all inputs and outputs as unmatched, then repeat the following steps in each of I iterations:

- Let $1 \leq i \leq I$ be the current iteration number. Let $k \leq N$ be the number of unmatched input-output pairs. Choose random matching $X_i(k)$ of this unmatched bipartite graph uniformly and randomly from the $k!$ possibilities.
- If $i < I$, retain the edges corresponding to $F_\eta[X_i(k)]$ and mark the nodes they cover as matched. If $i = I$, then retain all edges of $X_i(k)$.

This procedure yields a complete matching with N edges.

Merge procedure

`Merge` runs on two matchings, M_1 and M_2, and matching $\tilde{M}$. We describe **`Merge`** as follows.

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

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

- Let v be an unmarked input node. Set path $P_v = \emptyset$. Let $\psi(P_v)$ denote the weight of path P_v. Initially, set $\psi(P_v) = 0$.
- Let $(v, w) \in M_1$. Add (v, w) to P_v, and set $\psi(P_v) = \psi(P_v) + \psi(v, w)$.
- Let $(w, u) \in M_2$. Add (w, u) to P_v, and set $\psi(P_v) = \psi(P_v) - \psi(w, u)$.
- If $u = v$, stop. Or else, repeat the first three steps with u in place of v, and update P_v accordingly.
- 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 $\tilde{M} = \tilde{M} \approx M_1(P_v)$. Or else, let $\tilde{M} = \tilde{M} \approx M_2(P_v)$.
- If any node q is unmarked, start from the first step with q in place of v.

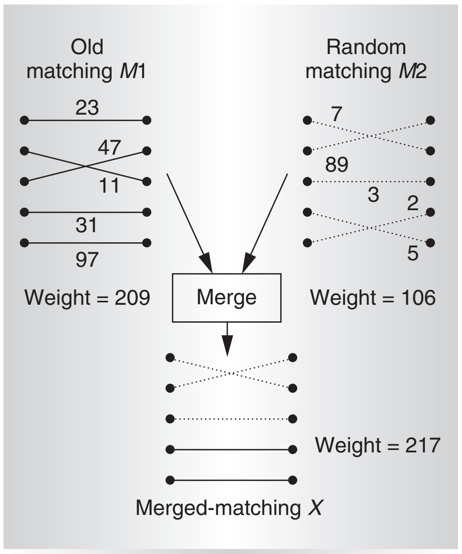

Figure 5. Merging example for matching M_1 and M_2. The weight of the final matching is always greater than or equal to the maximum weight of M_1 and M_2.

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

Figure 5 shows an example merging of matchings M_1 and M_2.

Stability properties

We assert that Laura is a stable algorithm—that is, it achieves 100 percent throughput under any admissible traffic patterns. Although we have proved this theorem, we omit the proof here because of space limitations.

Runtime

Laura's worst case running time 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 algorithm's runtime is $O(N \log^2 N)$. This is quite low compared to the runtime of $O(N^3)$ for MWM, $O(N^{2.5})$ for maximum-size matching, and for all other approximations thus far proposed. Max-Laura does extra work to make the matching maximal. If l of N nodes remain unmatched, then a simple algorithm to obtain

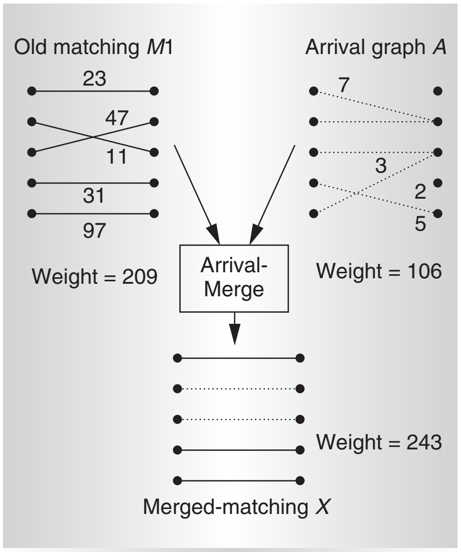

Figure 6. Arrival-Merge example for matching M_1 and arrival graph A. The final weight is always greater or equal to the weight of M_1.

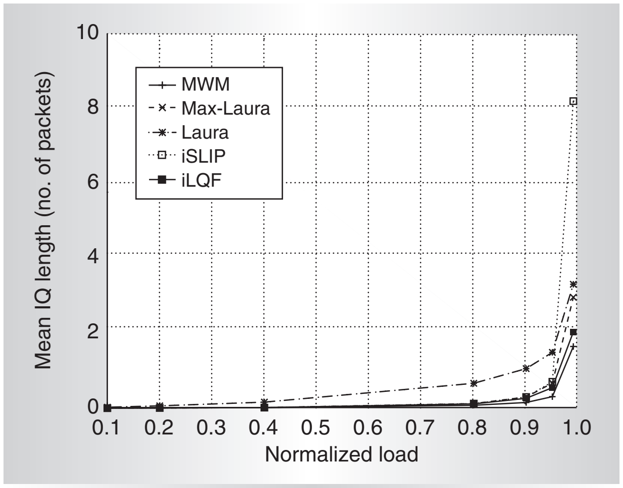

Figure 7. For uniform traffic, all the algorithms considered are well behaved (they do not exhibit long delays).

a maximal matching takes a worst-case time of $O(l^2)$. If l is small, this time is negligible compared to that of other algorithms. From a simulation study, we find that, in most cases, $l << N$, which suggests that the additional work done by Max-Laura is negligible.

Robustness

We explored Laura's sensitivity to several parameters to understand its robustness as its complexity decreased. We studied the sensitivity to I, S, and $d_{\min}$, and we also studied a nonrandomized version of Laura. In all these cases, we always experienced delays comparable to those of the original version of Laura.

Serena

Serena is a variant of Laura that uses packet arrival times as a source of randomness. It also uses an innovative merging algorithm. Laura uses the randomization to obtain unknown heavily weighted edges with low complexity. Observe that an edge becomes heavily weighted if its corresponding queue receives many arrivals and few services. Hence, an algorithm can capture the randomness provided by arrivals and use it to find heavily weighted edges.

The basic version of Laura merges the past schedule with a randomly generated matching. In contrast, Serena considers the edges that received arrivals in the previous time slot and merges them with the past matching to obtain a higher-weight matching.

Consider the bipartite graph at time t in which edges are those revealed by the arrivals. Thus, edge e_{ij} is present in this arrival graph if and only if a packet arrived at input i and was destined for output j at time t. Figure 6 shows an example arrival graph. This graph has at most N edges (because there is at most one arrival per input in each time slot), each input node has at most a degree of 1, but an output can have a degree of up to N.

Serena merges the arrival graph with matching $M(t)$ used in time slot $t - 1$ to obtain the schedule at time t. Here, we do not discuss the `Arrival-Merge` procedure in detail; but it is a simple procedure, loosely described as follows. For each output in the arrival graph that has multiple incident edges, choose the heaviest weighted edge and discard the other edges. This process results in a subgraph in which some input-output pairs are matched and others aren't. We arbitrarily match all unmatched input-output pairs. Serena merges the resulting matching with $M(t)$ to obtain schedule $S(t)$ at time t.

More formally, we describe Serena as follows:

- Let $M(t)$ be the matching used at time $t - 1$.
- Let $A(t) = A_{ij}(t)$ denote the arrival graph, where $A_{ij}(t) = 1$ indicates arrival, and $A_{ij}(t) = 0$ indicates otherwise.

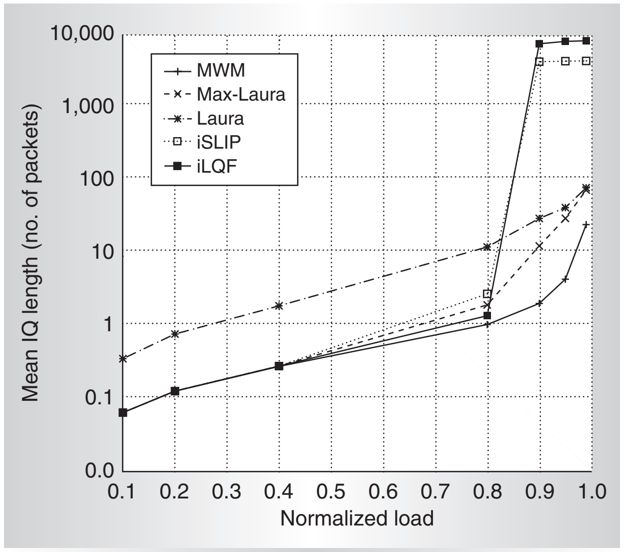

Figure 8. For diagonal traffic, Laura and Max-Laura can reach the same throughput as MWM. Max-Laura improves on Laura's performance because it is maximal.

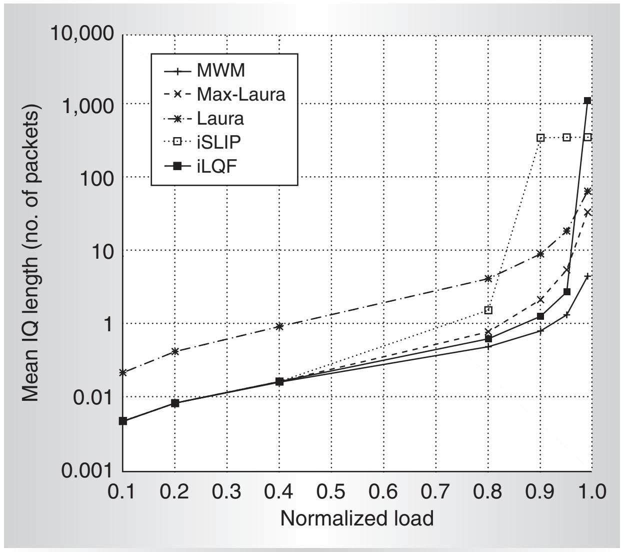

Figure 9. For log-diagonal traffic, Laura and Max-Laura outperform the other approximating algorithms under high load.

- Let $S(t)$ = **`Arrival-Merge`**$[M(t), A(t)]$, where **`Arrival-Merge`** is a special procedure, which we describe using the example in Figure 6.
- Use $S(t)$ as the schedule, and let $M(t + 1) = S(t)$.

Serena is even simpler than Laura, because it does not need to generate a random matching in each time slot. We note that Serena is a self-randomized algorithm—it does not use any external randomization.

Performance study

Figures 7, 8, and 9 compare the performance of Laura and Max-Laura, using the settings in Table 1. The figures compare these algorithms with well-known algorithms iSLIP[7] and iLQF,[8] both using N iterations.

Laura shows long delays for low load, because it is not maximal. By making Laura maximal, Max-Laura has delays as short as those of MWM even for low loads. Laura and Max-Laura outperform the other approximating algorithms for high load and nonuniform traffic patterns.

Figure 10 compares the performances of Serena and Laura with only one stored matching. It shows that Serena's **`Arrival-Merge`** outperforms Laura's usual **`Merge`**, which uses the **`Random`** procedure. But **`Arrival-`**

Table 1. Simulation settings for Laura and Max-Laura.

Parameter	Symbol	Value
Random-matching probes	V	1
No. of stored matchings	S	2
No. of iterations	I	5
Selection factor	η	0.5

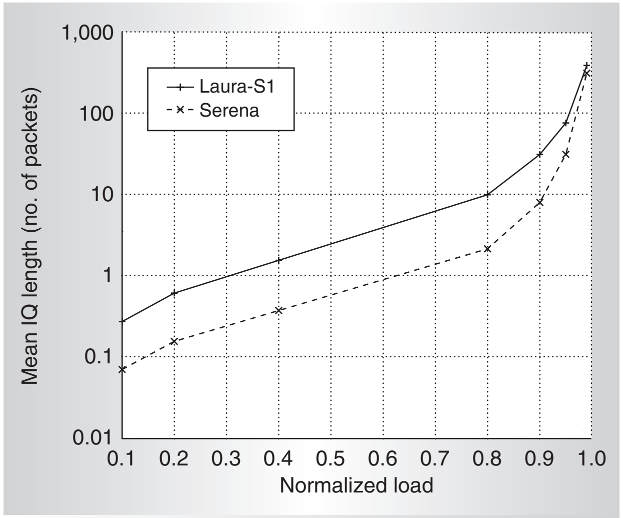

Figure 10. Comparison of Serena with Laura using one stored matching (S = 1) and under diagonal traffic. In this case, Serena's **`Arrival-Merge`** outperforms Laura's **`Merge`**.

`Merge` requires a somewhat more complex data structure. The choice of Serena or Laura should be based on the design tradeoffs associated with overall system performance.

Given the simplicity of the algorithms Laura and Serena, we would like to actually implement these algorithms in hardware in an actual switch. On the other hand, it will be interesting to theoretically prove that the delays of these algorithms are approximately close to that of the maximum-weight matching algorithm. MICRO

References

1. M. Mitzenmacher, *The Power of Two Choices in Randomized Load Balancing*, doctoral dissertation, Dept. of EECS, Univ. of California, Berkeley, 1996.
2. N. Vvedenskaya, R. Dobrushin, and F. Karpelevich, "Queueing System with Selection of the Shortest of Two Queues: An Asymptotic Approach," *Problems of Information Transmission*, Kluwer Academic Publishers, Dordrecht, The Netherlands, vol. 32, 1996, pp. 15-37.
3. K. Psounis and B. Prabhakar, "A Randomized Web-Cache Replacement Scheme," *IEEE Infocom 2001*, IEEE Press, Piscataway, N.J., 2001.
4. R. Motwani and P. Raghavan, *Randomized Algorithms,* Cambridge Univ. Press, Cambridge, UK, 1995.
5. N. McKeown, V. Anantharan, and J. Walrand, "Achieving 100% Throughput in an Input-Queued Switch," *Proc. 15th Ann. Joint Conf. of the IEEE Computer and Comm. Societies* (Infocom 96), IEEE CS Press, Los Alamitos, Calif., 1996, pp. 296-302.
6. J. Dai and B. Prabhakar, "The Throughput of Data Switches with and without Speedup," *IEEE Infocom 2000*, IEEE Press, Piscataway, N.J., 2000, pp. 556-564.
7. N. McKeown, "iSLIP: A Scheduling Algorithm for Input-Queued Switches," *IEEE Trans. Networking*, vol. 7, no. 2, Apr. 1999, pp. 188-201.
8. N. McKeown, *Scheduling Algorithms for Input-Queued Cell Switches,* doctoral dissertation, Dept. of EECS, Univ. of California, Berkeley, 1995.
9. M.M. Ajmone et al., "RPA: A Flexible Scheduling Algorithm for Input Buffered Switches," *IEEE Trans. Communications*, vol. 47, no. 12, Dec. 1999, pp. 1921-1933.
10. H. Duan et al., "A High Performance OC12/OC48 Queue Design Prototype for Input Buffered ATM Switches," *INFOCOM 97: 16th Ann. Joint Conf. of the IEEE Computer and Comm. Societies* (Infocom 97), IEEE CS Press, Los Alamitos, Calif., 1997, pp. 20-28.
11. L. Tassiulas, "Linear Complexity Algorithms for Maximum Throughput in Radio Networks and Input Queued Switches," *1998 IEEE Infocom*, IEEE Press, Piscataway, N.J., 1998, pp. 533-539.

Devavrat Shah is a PhD candidate in the Department of Computer Science at Stanford University. His research interests include the design, analysis, and implementation of network algorithms. Shah has a bachelor's degree in computer science and engineering from the Indian Institute of Technology, Bombay.

Paolo Giaccone is a PhD student in the Electronics Department at Politecnico di Torino, Italy. His research interests include scheduling algorithms for input-queued switches. Giaccone has a DrIng in telecommunications engineering from Politecnico di Torino. He is a student member of the IEEE.

Balaji Prabhakar is an assistant professor of electrical engineering and computer science and a Terman Fellow at Stanford University. He is also a fellow of the Alfred P. Sloan Foundation. His research interests include network algorithms (especially for switching, routing and quality of service), wireless networks, Web caching, network pricing, information theory, and stochastic network theory. Prabhakar has a PhD from the University of California at Los Angeles. He has received a Career award from the National Science Foundation and the Erlang Prize from the Informs Applied Probability Society.

Direct questions and comments about this article to Devavrat Shah, Dept. of Computer Science, Stanford Univ.; 270 Packard EE Building, 350 Serra Mall, Stanford, CA 94305-9510; devavrat@cs.stanford.edu.