Time-Encoding Scheduling

Li-yen Chen and Petar Momčilović

***Abstract*—This paper develops a novel scheduling mechanism for packet switches based on a minimal set of hardware components. The standard model of computation implemented by digital logic is replaced by a model based on time encoding. This new model allows for distributed computation with low hardware complexity. A distributed switch scheduling algorithm utilizing time encoding is shown to deliver performance comparable to centralized algorithms under uniform traffic. Exploiting a connection between switch scheduling and interval packing, we argue that the distributed nature of the algorithm limits the maximum relative load to $1 - e^{-2}$ under diagonal traffic. The stability of the algorithm can be improved by enabling reversibility in distributed decision making.**

I. Introduction

High performance routers and switches, being the core of various existing and proposed architectures, are likely to remain indispensable parts of future networks. This paper develops a novel model of computation that enables efficient operation of packet switches with distributed scheduling; the model yields a new architecture and scheduling algorithms. Scalability of packet switches has been considered in the literature. In short, there exists a tradeoff between the switch performance and hardware complexity, as well as between the complexity of data-plane hardware (crossbars, memory banks, etc.) and control plane hardware (control unit). However, the studies in this domain are dominated by architectures with either a significant number of switching components and/or a centralized controller. The assumption of system coordination is typically implied in those models. Our paper is not based on this assumption. Namely, we design a fully distributed scheduling scheme for packet switches. Distributed operation is enabled by introducing a novel model of computation, and thus eliminating the standard model based on bits, registers, machine cycles and all related hardware. The new model of computation utilizing time encoding is capable of drastically reducing the complexity of the control plane, since inter-port communication on a single piece of switching fabric is not required. While a number of alternative models of computation have been proposed (molecular, quantum, membrane, etc.), at present devices based on these models do not outperform those using the standard model. In contrast, the architecture we propose provides performance that is comparable with known centralized architectures.

Our model of computation is inspired by neurons. In particular, the relevant quantities for determining a feasible schedule are represented by the length of time intervals instead of conventional bits in registers; hence, the term time encoding. Computation required to determine a state of the switching fabric can be performed efficiently under such representation. We point out that the model is potentially extendable to other inherently decentralized systems such as wireless networks. Our focus on the switch design problem is dictated by two factors: (i) switch scheduling problem is well-defined, structured and important in practice, and (ii) the resulting architecture can be implemented physically.

The remaining of this paper is organized as follows. In the next section, we outline the basic switch model and briefly review relevant literature. Section III describes a novel model of computation based on time encoding, where time intervals are used to perform computations. In Section IV we discuss a distributed scheduling algorithm utilizing this model and its implementation. A reversible version of our algorithm is presented in Section V. Finally, concluding remarks can be found in Section VI.

II. Model

A. Virtual-Output-Queued switch

The Virtual-Output-Queued (VOQ) switch architecture has the following desirable properties: (i) it is based on a switching fabric operating at line speeds, while output-queued switches require hardware speedup; (ii) it does not suffer from performance degradation due to the head-of-line (HOL) blocking as in the case of input-queued switches; (iii) it can be implemented on a single switching fabric (crossbar); (iv) no reordering of packets is required. The reduced hardware requirements and improved performance come at the cost of increased control complexity. Hence, the key performance metrics (throughput, delay, etc.) crucially depend on the employed scheduling algorithm. In particular, it has been shown that a greedy algorithm performs poorly under certain traffic scenarios [22].

The structure of an $N \times N$ VOQ switch is shown in Fig. 1. Packets arriving to input i with destination j are enqueued in the virtual output buffer (i, j). We assume that the buffer is of infinite size, that time is slotted, and that every packet is of fixed size such that it takes exactly one time slot to transmit a packet. The number of packets in queue (i, j) at time slot t is denoted by $Q_{i,j}(t)$. Let $A_{i,j}(t)$ be the number of packets arriving to queue (i, j) at time t. The queue occupancies evolve in time according to

$$Q_{i,j}(t+1) = Q_{i,j}(t) - S_{i,j}(t) + A_{i,j}(t), \qquad (1)$$

where $S_{i,j}(t) = 1$ if queue (i, j) is non-empty and is served at time t, i.e., a packet is switched from input i to output j; otherwise $S_{i,j}(t) = 0$. Event $\{S_{i,j}(t) = 1\}$ indicates that the

This work was supported by the National Science Foundation under Grant CNS-0643213.

L. Chen is with the network virtualization team at Microsoft Corp., Redmond, WA 98052 USA (e-mail: liche@microsoft.com).

P. Momčilović is with the Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: petar@ise.ufl.edu).

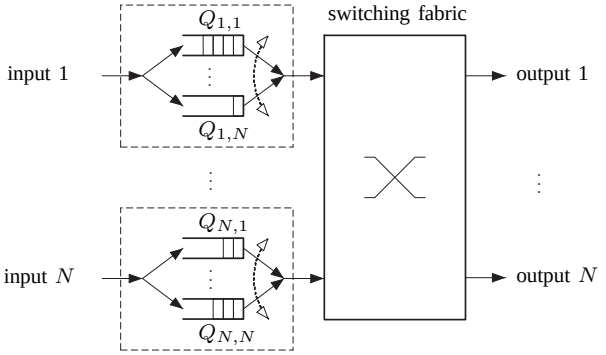

Fig. 1. The structure of a Virtual-Output-Queued switch. Packets are enqueued at inputs based on their destinations. Each input/output can be connected to at most one output/input in every time slot.

switching fabric is configured to a state such that one packet in queue (i,j) is switched, and, thus, $S_{i,j}(t)$ depends on the scheduling algorithm. The switching fabric is physically constrained to transmit at most one packet for each input/output port in a time slot, i.e.,

$$\sum_{i=1}^{N} S_{i,j}(t) \leq 1 \quad \text{and} \quad \sum_{j=1}^{N} S_{i,j}(t) \leq 1. \tag{2}$$

A scheduling algorithm determines the value of $\{S_{i,j}(t)\}$ subject to (2). Hence, in view of (1), the impact of scheduling algorithm on the evolution of queue occupancy processes is only through the values of $\{S_{i,j}\}$.

In the rest of this paper, we assume that the arrival processes satisfy a strong law of large numbers, i.e., with probability 1,

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} A_{i,j}(t) = \lambda_{i,j}, \quad i,j = 1,\ldots,N, \tag{3}$$

where $\lambda_{i,j}$ is the traffic intensity at virtual queue (i,j). We say that a scheduling algorithm is *rate stable* if, with probability 1,

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} S_{i,j}(t) = \lambda_{i,j}, \quad i,j = 1,\ldots,N,$$

for any arrival process satisfying (3). The input traffic is called *admissible* if

$$\lambda_{\cdot,j} = \sum_{i=1}^{N} \lambda_{i,j}(t) < 1 \quad \text{and} \quad \lambda_{i,\cdot} = \sum_{j=1}^{N} \lambda_{i,j}(t) < 1. \tag{4}$$

It is straightforward to show that no scheduling algorithm is rate stable under non-admissible traffic ($\lambda_{\cdot j} > 1$ or $\lambda_{i\cdot} > 1$ for some i, j). In that case, $Q_{i,j}(t)$ increases over time without a bound regardless of the scheduling algorithm for some i and j.

B. Maximum-Weight-Matching scheduling

Maximum-Weight-Matching (MWM) algorithm is rate stable under any admissible traffic pattern and is known to provide high throughput and low delay [8], [11], [18], [15], [29]. We sketch the operation of MWM below. The state of the switch can be described by a bipartite graph, with vertices representing input and output ports, and edges representing the corresponding virtual buffers. In each time slot t, edge (i,j) is assigned a weight $W_{i,j}(t)$ that measures the congestion level of the corresponding queue, e.g., $W_{i,j}(t) = Q_{i,j}^{\beta}(t)$, $\beta > 0$ [2], [26]. The MWM algorithm selects a matching (set of independent input-output pairs) with the highest sum of weight and schedules transmissions according to the matching. When $W_{i,j}(t) = 1_{\{Q_{i,j}(t)>0\}}$, the algorithm reduces to the Maximum-Size-Matching (MSM) algorithm. In [42], [26] it was shown that the MWM algorithm is rate stable with Bernoulli arrival processes when no input or output port is overloaded (admissible traffic); the result was extended for more general arrival processes in [8]. MWM-like algorithms were also argued to perform well when applied in the networks of constrained queues [40] and switches [24]. Furthermore, the MWM algorithm with appropriately chosen weight functions performs optimally under the heavy load scenario not only in switches [35], [38] (the cross-bar switch considered here is a special case of the generalized switch examined in [38]), but also in stochastic processing networks [7], [3] (optimality is considered with respect to holding cost).

Nonetheless, computing the maximum weight matching at line speeds is a challenge. Existing algorithms [11], [18], [15] are either of significant complexity or poor scalability, i.e., requiring a large number of arithmetic operations for a large number of input/output ports. Hence, a number of practical algorithms were developed, including iSLIP [25], MUCS [12] and RPA [23]. These algorithms do not attempt to approximate the MWM algorithm explicitly and are inferior to the MWM when the input traffic is not uniform [17] (the definition of uniform traffic can be found in Section IV-C). On the other hand, algorithms based on MWM approximations were also considered, see e.g. [41], [16], [9]. They utilized the observation that queue occupancies exhibit correlation in time, and hence, the weight of a matching does not change significantly over a small time interval. Recently, an algorithm based on an auction is proposed in [4].

Distributed switch scheduling on a single crossbar poses additional challenges. Iterative algorithms such as iSLIP [25] and the bidding algorithm [4] can be considered distributed but they require multiple rounds of "information exchange". Namely, input and output ports communicate between each other in order to establish a feasible schedule. Note that the problem of switch scheduling can be viewed as an instance of a general problem of scheduling on interference graphs. Interference graphs were also considered in the context of scheduling in wireless networks. In such networks information exchange (message passing) is even costlier since control packets utilize the same resource (bandwidth) that needs to be allocated (scheduled). Algorithms that require a large number of rounds of message exchanges are capable of supporting switch operation at limited speeds only since a non-negligible amount of time is needed for rounds to take place. On the other hand, schemes with simple implementations such as maximal scheduling (single round of scheduling) deliver only a fraction of the possible throughput [5], [34]. Distributed algorithms with low-complexity considered in the literature include local back-pressure policy [40] and the Longest-Queue-First (LQF) policy [10]. For further studies of distributed scheduling in

wireless networks see [29], [14], [37] and references therein. The extension of our basic algorithm, the reversible algorithm described in Sect. V, is conceptually closest to the distributed CSMA algorithm proposed in [19]; in particular, even though the algorithms are different, the structures of underlying Markov chains bear resemblance.

C. Algorithm comparisons

Although it is desirable to have a scheduling algorithm with a higher relative throughput, this oftentimes comes at the cost of larger time complexity. When comparing the performance of scheduling policies for the single crossbar switch, both the quality and running time of decision making need to be taken into account. In particular, if an algorithm achieves relative throughput $\rho \in [0, 1]$ and takes τ seconds to determine a schedule, then the maximum theoretic throughput is at most $\rho\tau$ cells/s. For example, consider two scheduling algorithms $\mathcal{C}$ and $\mathcal{D}$, achieving 100% and 85% of the relative throughput, respectively. If it takes 10^{-8} s for $\mathcal{C}$ and 10^{-10} s for $\mathcal{D}$ to compute a schedule, then the maximum theoretic throughput is at most 10^8 cells/s for $\mathcal{C}$ and $85 \cdot 10^8$ cells/s for $\mathcal{D}$. Therefore $\mathcal{D}$ significantly outperforms $\mathcal{C}$ due to a faster decision making. We note that when a pipeline architecture is considered, scheduling time may not be the only performance bottleneck [36]. However, such an architecture not only requires higher hardware complexity, but also implies longer delays which can result in throughput loss from the mismatch of the current state and delayed decision, especially under a fast-varying traffic.

III. Time-encoding computation

In this section, we discuss a model of computation based on time encoding. The standard model of computation is based on bits and elementary operations (e.g., summation comparison and multiplication) and is implemented in digital logic. Although some early computing machines were based on analog components, today digital circuits dominate the implementation of specialized and general-purpose processors. Nonetheless, alternative models of computation have received considerable attention (e.g., see [1], [30], [33]), most of which are yet to be physically implemented. On the other hand, the model of computation we propose can be integrated into the existing solid-state technology.

A. Time-encoding operations

In our model of computation, variables are represented with time intervals of possibly random lengths rather than deterministic bits. A time interval (and hence its length) can be defined by two impulses. Let X and Y be two variables. The following two basic operations can be utilized to implement time-encoding algorithms:

- *Minimum* ($\min\{X, Y\}$). Variables X and Y are represented by two constant time intervals with lengths X and Y, respectively. Define the new starting impulse as the beginning of the two aligned time intervals, and the new ending impulse as the earlier of the two ending impulses. Clearly, the length of the new time interval is $\min\{X, Y\}$.

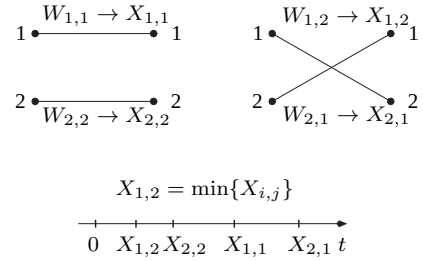

Fig. 2. Illustration for the motivating example. The switching fabric has only 2 possible states shown on the top. On the bottom is the time line that indicates the expiration of the timers. In this example, the timers are set at $t = 0$. At $t = X_{1,2}$, the first timer to expire is the one associated with input-output pair $(1, 2)$. Thus, in this instance the switch is configured to the matching with weight M_2.

- *Inverse sum* ($1/(X + Y)$). Variables X and Y are represented by two random-length time intervals. In particular, the length of the intervals are given by exponential random variables with means $1/X$ and $1/Y$; denote the lengths of these intervals by T_X and T_Y, respectively. The minimum of the two, $\min\{T_X, T_Y\}$, can be obtained by applying the minimum operator. Random variable $\min\{T_X, T_Y\}$ is exponentially distributed with mean $1/(X + Y)$. Hence, the output of the operator, $\min\{T_X, T_Y\}$, is only an approximation of the inverse sum, $1/(X+Y)$, but can be implemented in a distributed fashion.

B. A motivating example

Next, we illustrate how the basic time-encoding operations can be used with a motivating example. For simplicity, consider a 2×2 switch with input-output weights given by $[W_{i,j}]_{i,j=1,2}$. In this case there exist two matchings (switch configurations) with weights $M_1 = W_{1,1} + W_{2,2}$ and $M_2 = W_{1,2} + W_{2,1}$ (with corresponding connectivity patterns $\{(1, 1), (2, 2)\}$ and $\{(1, 2), (2, 1)\}$ as shown in Fig. 2). The MWM algorithm schedules a matching with the maximum weight. Given the standard model of computation based on digital logic (i.e., bits, registers, summations, comparisons, etc.), effectively a single entity is required in order to determine which matching has a higher weight. In particular, the entity needs to be aware of the values of $[W_{i,j}]_{i,j=1,2}$ so that M_1 and M_2 can be computed and compared.

Our distributed scheme using information encoded in timing is described next. The scheme is randomized with the following intuition: the heavier the matching the more likely it is scheduled. In the first step each input-output weight is randomized. In particular, $W_{i,j}$ is replaced with $X_{i,j}$ where $X_{i,j}$ is exponentially distributed with $\mathbb{E}X_{i,j} = 1/W_{i,j}$. This is the step that enables distributed summation. Observe that the randomization can be implemented in a distributed fashion since each $X_{i,j}$ depends only on the corresponding value of $W_{i,j}$. In the second step, at some fixed time, say $t = 0$, each input-output pair (i, j) (virtual buffer) sets a timer with an expiration time $X_{i,j}$. Upon the expiration of timer (i, j), an attempt is made to connect input i and output j. Such a connection is feasible only if both input i and output j have

not yet been connected to some other ports. The algorithm terminates when all finite timers expire. We call the algorithm distributed since each virtual queue attempts a transmission based on local information only. Note that in the 2×2 case the matching is scheduled whenever one of its two input-output pairs is scheduled first (equivalently, a timer expires first). For example, matching with weight M_1 is scheduled if and only if $\min\{X_{11}, X_{22}\} < \min\{X_{12}, X_{21}\}$. The key fact is that $\min\{X_{11}, X_{22}\}$ and $\min\{X_{12}, X_{21}\}$ are exponentially distributed with means $1/M_1$ and $1/M_2$, respectively. Effectively, we use the comparison of $\min\{X_{11}, X_{22}\}$ and $\min\{X_{12}, X_{21}\}$ (fully distributed based on timers) as a proxy for the comparison between M_1 and M_2. A straightforward computation shows that the two matchings are chosen with probabilities $M_1/(M_1 + M_2)$ and $M_2/(M_1 + M_2)$.

IV. Distributed scheduling

The centralized implementation of the MWM algorithm requires the presence of a control unit in the switch architecture. Previous approaches to eliminate the control unit come at cost of increasing the complexity of the switching hardware. Informally, the idea is to shape the input processes (make the input traffic uniform) so that a simple scheduling algorithm can be employed, e.g. see the load-balanced router architecture [21], [20]. The increased complexity stems from the additional hardware required to reorder the packets (in order to restore the original sequencing) prior to their transmission to output ports. In contrast, our approach aims at developing a distributed scheduling algorithm (i.e. no control unit is required) that maintains the original order of packets within the switch. The key idea behind our architecture is to enable distributed summations so that matching weights can be computed without explicit information exchange between ports.

A. Basic algorithm

In this subsection, we describe a randomized algorithm $\mathcal{A}(W)$. The argument W refers to a function that is used to assign weights to input-output pairs. For example, if $W = Q^{(\beta)} = [Q_{i,j}^{\beta}]$ (as proposed originally in [2]), then at time slot t the weight of the edge (i, j) in the corresponding input-output bipartite graph is given by $Q_{i,j}^{(\beta)}(t)$. We use $[T_{i,j}(t)]$ to denote an $N \times N$ matrix that is independent of the arrival processes. Elements of each matrix are independent and exponentially distributed random variables with unit mean; the matrices for different values of t are independent. The algorithm operates as follows.

ALGORITHM $\mathcal{A}(W)$ [27]
For each time slot t:

1. Define X as an $N \times N$ matrix with elements
$$X_{i,j} := T_{i,j}(t)/W_{i,j}. \quad (5)$$
2. Match input i to output j such that
$$(i, j) = \arg\min X_{k,l}; \quad (6)$$
delete the ith row and jth column from X.
3. Repeat the preceding step until X is empty or $X_{i,j} = \infty$ for all i, j.

It is clear that $\mathcal{A}(W)$ implements LQF scheduling with "randomized queue length" given by $X_{i,j}^{-1} = W_{i,j}/T_{i,j}$. The reason for not defining $\mathcal{A}(W)$ as simply a randomized LQF algorithm is that the above description of the algorithm allows for a distributed implementation (see the next subsection). Such an implementation is feasible due to the $\arg\min$ operator in (6) – the LQF algorithm requires the $\arg\max$ operator. The algorithm $\mathcal{A}(W)$ with an appropriately chosen weight function has the following desired characteristics [25]:

- High throughput: in the following subsections we argue that the algorithm remains stable under relatively high traffic loads.
- Starvation free: $\mathcal{A}(W)$ eventually serves all virtual output queues as long as the total backlog is not infinite and W is proportional to the queue lengths; there exists a positive probability for any of the non-empty queues to be scheduled.
- Simple implementation: aside from the source of randomness, the algorithm is straightforward to implement. Conceptually, operation of each input-output pair (virtual queue) remains the same, regardless of the size of the switch.

B. Distributed implementation

Suppose that the source of randomness is available at each input port; that is, for each time slot, unit mean exponential random variables $[T_{i,j}]$ are available at the input ports. By adjusting the intensities of these random variables based on $W_{i,j}$ (e.g., based on $Q_{i,j}(t)$), one obtains $[X_{i,j}]$. At a fixed time, each input-output pair (i, j) sets a timer that expires in $X_{i,j}$ time units; note that these units do *not* correspond to time slots needed to switch a packet (cell) from an input to an output. Upon the expiration of timer (i, j), input i is matched to output j unless either of them is already matched. This process continues until all timers with $X_{i,j} < \infty$ expire. An example of how the algorithm operates for a 3×3 switch is shown in Fig. 3. The timers are set at time $t = 0$; as they expire, attempts are made to connect the corresponding input and output ports. Upon the completion of the scheduling phase (in this case the input ports 1, 2 and 3 are connected to the output ports 2, 1 and 3, respectively), the cell transfer phase begins.

Note that the implementation is distributed whenever $W_{i,j}$ is only a function of the state of the virtual queue (i, j), i.e., the timers are set with local parameters only. However, the complexity of implementing the required source of randomness needs to be considered. There exists at least two possible approaches to obtain randomness: (i) external – a physical device (e.g. photon-based) auxiliary to the switching fabric can serve as a source of randomness (Poisson process); these devices can potentially operate at very high speeds; and (ii) internal – randomness can be extracted from packet

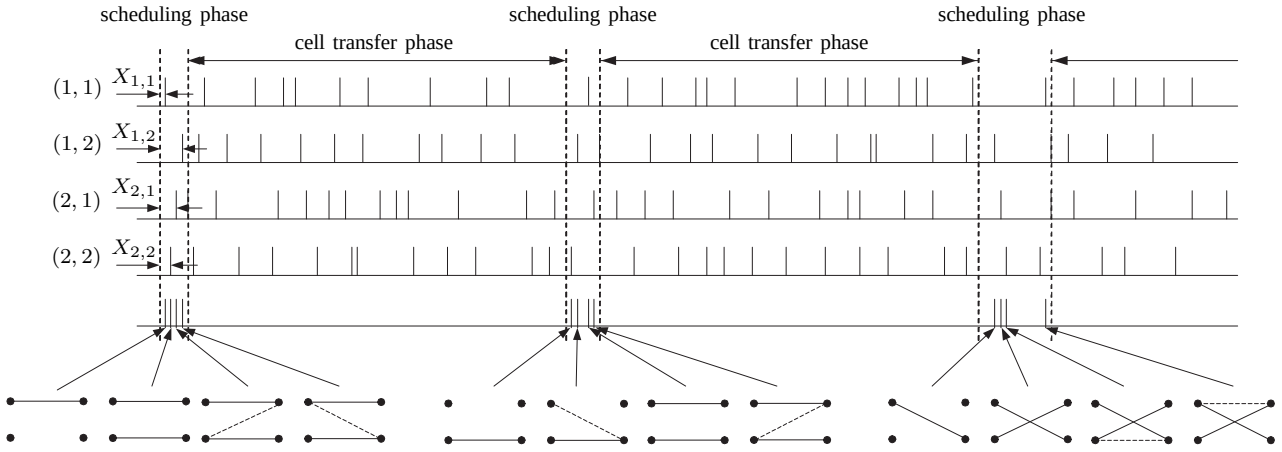

Fig. 4. Conceptual operation of a 2×2 switch under $\mathcal{A}(W)$. Poisson processes are generated at each virtual buffer (input-output pair) with rates proportional to the occupancy continuously over time; these processes serve as the required timers during scheduling phases. At the beginning of each scheduling phase, all input-output ports are disconnected. As impulses arrive (timers expire), the corresponding feasible connections are established. The configuration of the switching fabric after each arrival of the Poisson processes is shown on the bottom of the graph. Once the fabric is configured, a cell transfer phase starts.

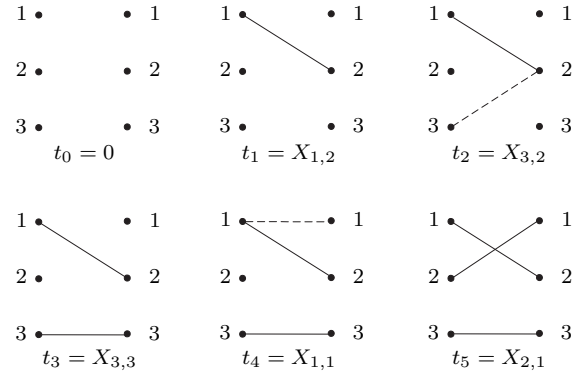

Fig. 3. An example of how a matching is built in the case $X_{1,2} < X_{3,2} < X_{3,3} < X_{1,1} < X_{2,1} < \infty$ and all other $X_{i,j}$'s are equal to ∞. The first timer expires at $t_1 = X_{1,2}$, the second at $t_2 = X_{3,2}$ and so on. When a timer expires, the input is connected to the output only if such a connection is feasible. Non-feasible connections are shown with dashed lines.

payloads, e.g., see [13]. We note that lasers and photon-counting semiconductor detectors can be used to implement Poisson timers [43]. Intensities of external Poisson processes (sequences of impulses) at virtual buffers are modulated according to buffer occupancies. By the memoryless property of Poisson processes, an exponentially-distributed time interval can be defined by a single impulse, namely, the ending impulse. Then, the $\arg\min$ of a set of variables is determined by the first impulse from the corresponding set of Poisson processes. Thus, the algorithm can be implemented through the two operations described in Sect. III-A. Moreover, in order to simplify the source of randomness, such impulses are generated continuously as shown in Fig. 4. The switch operates in two alternating phases: scheduling and cell transfer; the length of the cell transfer phase is relatively longer. Note that in practice, each scheduling phase is of fixed length, and there is a positive probability that the some impulses do not arrive before the scheduling phase ends. We adjust the arrival rate to be large compared to the length of the scheduling phase, hence approximating algorithm $\mathcal{A}$. In Fig. 4 we illustrated the conceptual operation of the 2×2 switch. At the beginning of a scheduling phase, a sequence of impulses is generated at each virtual buffer. As impulses are generated (equivalently timers expire) from the Poisson processes, the switching fabric is configured accordingly. The resulting configurations are shown on the bottom of the figure. Once the configuration is determined, the cell transfer phase starts. Upon completion of a cell transfer phase, the switching fabric is reset, i.e., all input and output ports are disconnected, and the scheduling phase for the next time slot begins.

Finally, we point out that one can potentially implement the LQF algorithm by setting $T_{i,j}$'s in the definition of $\mathcal{A}(W)$ to constants rather than of unit-rate random variables. However, in that case one will need to implement timers that allow for synchronization of initial impulses. In addition, producing time intervals of precise lengths poses technical challenges.

C. Performance

The distributed nature of the proposed architecture and the scheduling algorithm results in a low complexity of the control plane. In this subsection, we evaluate the performance of $\mathcal{A}$. Inefficiencies of $\mathcal{A}$ are due to the fact that inverse summation is performed only approximately (see Sect. III-A), and, thus, the maximum weight matching is evaluated approximately as well. First, we consider the uniform traffic pattern defined by arrival rates $\lambda_{i,j} = \rho/N$, $1 \le i, j \le N$, where $\rho \in [0, 1)$ is the relative traffic load. A preliminary version of the following result appeared in [27].

Theorem 1: Algorithm $\mathcal{A}(Q^{(\beta)})$, $\beta > 0$, is rate stable under any admissible traffic in a 2×2 switch. Moreover, $\mathcal{A}(Q^{(\beta)})$, $\beta > 0$, is rate stable under uniform admissible input traffic in a switch of an arbitrary size.

Proof: See Appendix A. ■

Simulation results indicate that, in $N \times N$ switch, $\mathcal{A}$ is rate stable under a broad class of admissible traffic patterns that includes uniform traffic. The difficulty of characterizing such a

class is due to the high dimensionality of input traffic patterns. In order to explore performance limits of $\mathcal{A}$, we consider an extreme case (worst-case), i.e. the diagonal traffic pattern. It is parameterized by $\alpha \in [0,1]$, and defined by

$$\lambda_{i,j} = \begin{cases} \alpha\rho, & j = i, \\ (1-\alpha)\rho, & j = (i \bmod N) + 1, \\ 0, & \text{otherwise;} \end{cases} \tag{7}$$

parameter $\rho \in [0,1)$ is the relative load for each input and output. Under this traffic pattern, the queue occupancy process is unstable under $\mathcal{A}(Q^{(\beta)})$ when $\rho > \rho^*$, where ρ^* denotes the value of the critical relative load. The inefficiency arises due to the distributed nature of the algorithm. Note that $\rho^* < 1$ when empty queues are served with positive probabilities in the long run. We illustrate this point with the following example. Consider a 3×3 switch under diagonal traffic with $\alpha = 1/2$ and $\rho = 1$. In the long run, the switch should be configured into one of the two following service matrices:

$$m_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad m_2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \tag{8}$$

with probability 1. Any other configuration inevitably "wastes" one unit of service. For example, if connection $(1,1)$ is made, it must follow that queues $(2,2)$ and $(3,3)$ will also be served in the same time slot. However, since $\mathcal{A}(Q^{(\beta)})$ is distributed, the configuration for the remaining input/output ports (2 and 3) are not constrained by the established connection that corresponds to $(1,1)$. Hence, when all queues with positive arrival rates are large, the scheduling algorithm selects a service matrix other than m_1 and m_2, such as

$$m_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \tag{9}$$

with a positive probability, where only two units of service occur. Note that when the switching fabric is configured according to the above service matrix, no other input-output pair with positive arrival rate can be scheduled concurrently. Although the input-output pair $(3,2)$ can be scheduled in addition to the scheduled pair $(1,1)$ and $(2,3)$, it has zero arrival rate and hence corresponds to an empty queue for the given traffic pattern – service of queue $(3,2)$ does not contribute to an actual departure of a packet.

The following proposition establishes a performance limit of $\mathcal{A}(Q^{(\beta)})$ under the diagonal traffic pattern.

Proposition 1: Consider algorithm $\mathcal{A}(Q^{(\beta)})$, $\beta > 0$, under diagonal traffic with $\alpha = 1/2$. The algorithm is not rate stable for

$$\rho > 1 - \left(\sum_{r=0}^{2N-1} \frac{(-2)^r}{r!} + \frac{1}{2} \frac{(-2)^{2N}}{(2N)!} \right) \to 1 - e^{-2}, \tag{10}$$

as the number of input/output ports $N \to \infty$.

Proof: See Appendix B. ∎

Results of numerical comparisons of $\mathcal{A}(Q)$, MWM, LQF, and APSARA [16] (a heuristic algorithm based on MWM with a smaller complexity; ASPARA selects between the matching in the previous time slot and a matching from a random walk on the graph of all matchings) under uniform and diagonal traffic are shown in Fig. 5 and Fig. 6, respectively. The arrivals form independent Bernoulli processes, and the number of input/output ports is $N = 32$. For a range of values of ρ, two quantities,

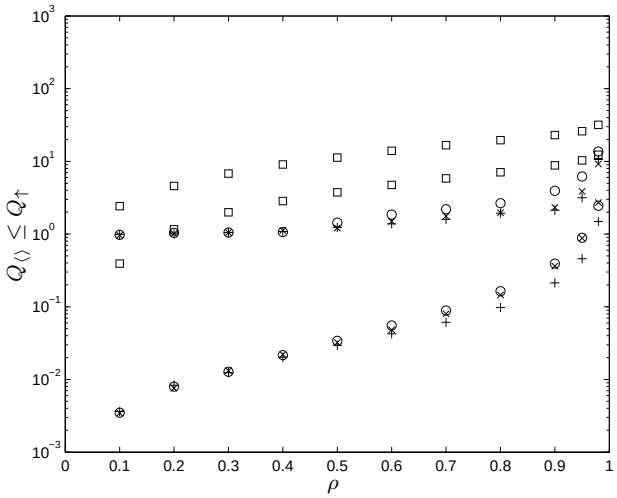

Fig. 5. For a switch with $N = 32$, the expected average virtual output queue length $Q_{\langle\rangle}$ and expected maximum virtual output queue length $Q_{\uparrow}$ for the MWM (+), APSARA (□), LQF (×), and $\mathcal{A}(Q)$ (○) algorithms under uniform traffic.

$$Q_{\langle\rangle} = \frac{1}{N^2} \sum_{i,j=1}^{N} \mathbb{E} Q_{i,j}(t) \tag{11}$$

and

$$Q_{\uparrow} = \mathbb{E}\left[\max_{i,j} Q_{i,j}(t) \right],$$

are estimated under the four algorithms. It is clear that $Q_{\langle\rangle} \leq Q_{\uparrow}$, and, thus, we do not label data points in the figures. Note that for the diagonal traffic pattern one can alternately normalize $Q_{\langle\rangle}$ by $2N$ instead of N^2 as in (11), since only $2N$ virtual queues receive packets. Given that we consider only relative performance of algorithms, the normalization does not effect our comparison.

As seen in Fig. 5, the algorithm $\mathcal{A}(Q)$ performs competitively under uniform traffic for a wide range of ρ, when compared to *centralized* switch scheduling algorithms. Numerical results for diagonal traffic with $\alpha = 1/2$ and $\alpha = 2/3$ are shown in Fig. 6. As observed, the algorithm fails to stabilize the switch for high values of relative load ρ as indicated in Proposition 1. The LQF algorithm also fails to stabilize the switch under diagonal traffic as argued in [10] (the sufficient local pooling condition is not satisfied in this case). For $\alpha = 1/2$ an estimate of the critical ρ^* is given in Proposition 1. An upper bound of the critical relative load for $\alpha = 2/3$ can be evaluated numerically as described in Appendix C. In particular, $\rho^* < 0.8744\ldots$ for $\alpha = 2/3$ and $\mathcal{A}(Q)$. The following section describes a scheduling algorithm that addresses the instability.

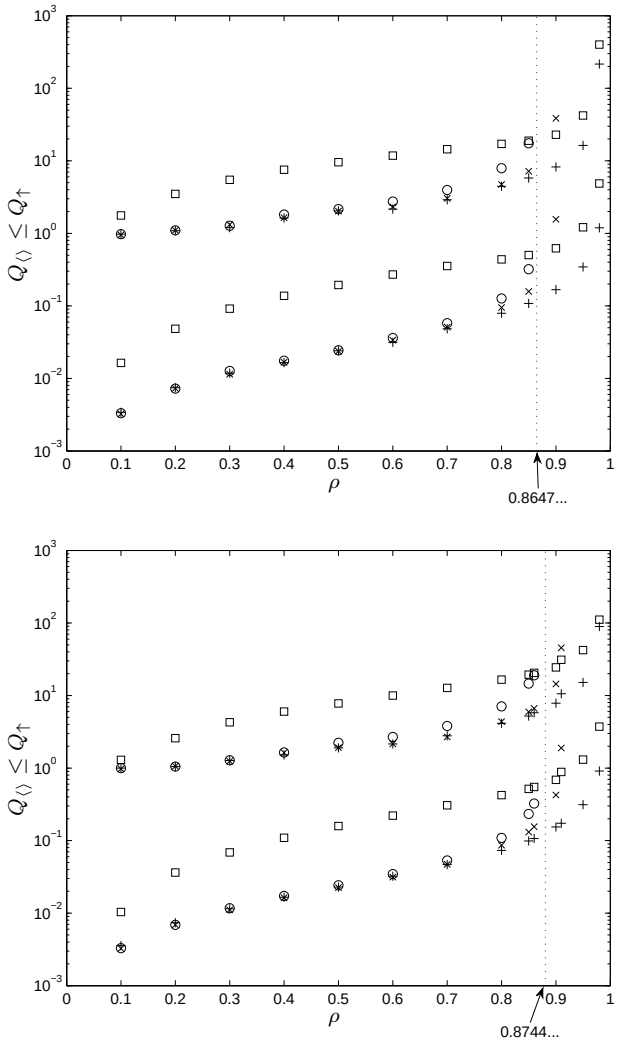

Fig. 6. For a switch with $N = 32$, the expected average virtual output queue length $Q_{\langle\rangle}$ and expected maximum virtual output queue length $Q_{\uparrow}$ for the MWM (+), APSARA (□), LQF (×), and $\mathcal{A}(Q)$ (○) algorithms under diagonal traffic with $\alpha = 1/2$ (top) and $\alpha = 2/3$ (bottom). The corresponding maximum ρ's for $\mathcal{A}(Q)$ are also shown.

V. Reversible algorithm

Algorithm $\mathcal{A}$ makes irreversible scheduling decisions; once a connection between two ports is established, these ports remain connected until the next scheduling phase, regardless of consequent scheduling decisions. In this section, we demonstrate that enabling reversibility in decision making potentially improves the scheduling performance.

A scheduling algorithm $\mathcal{B}_{(\lambda,\mu,\tau)}(W)$, where W is the set of weights, operates during a fixed length (τ time units) scheduling phase. Similar to algorithm $\mathcal{A}(W)$, $\mathcal{B}_{(\lambda,\mu,\tau)}(W)$ attempts to connect eligible input-output pairs upon arrivals in the corresponding Poisson processes. However, all connected pairs are subject to disconnections based on arrivals in rate-μ Poisson processes. In particular, for an unmatched pair (i, j), a connection is attempted according to a Poisson process with rate $\lambda W_{i,j}$; a matched pair (k, l) is disconnected after an exponentially distributed time interval with rate μ. When $\mu = 0$ and $\lambda \to \infty$, the algorithm reduces to $\mathcal{A}(W)$ (since τ is finite). The state of the switching fabric is fixed at the end of the scheduling phase for the duration of the cell transfer phase. The operation of $\mathcal{B}_{(\lambda,\mu,\tau)}$ is formally described below. Here, for $t \in [0, \tau]$, $[T_{i,j}(t)]$ is an $N \times N$ matrix with elements being i.i.d. unit-rate exponentially distributed random variables; the matrices are independent for different values of t. Let $\mathcal{X}$ be the set of all feasible fabric configurations:

$$\mathcal{X} = \left\{ \mathcal{I} \subseteq [1, N]^2 : \sum_{i=1}^{N} 1_{\{(i,j)\in\mathcal{I}\}} \le 1,\ \sum_{j=1}^{N} 1_{\{(i,j)\in\mathcal{I}\}} \le 1 \right\},$$

where $[1, N] \equiv \{1, 2, \ldots, N\}$. Without loss of generality, assume that the scheduling phase begins at $t = 0$.

ALGORITHM $\mathcal{B}_{(\lambda,\mu,\tau)}(W)$

1. Initialization:

$$\begin{aligned} &X_{i,j} := T_{i,j}(0)/(\lambda W_{i,j}),\ 1 \le i, j \le N, \\ &Y_{i,j} := \infty,\ 1 \le i, j \le N, \\ &\mathcal{S} := \emptyset. \end{aligned} \tag{12}$$

2. While $t \le \tau$:

$$\begin{aligned} (i, j) &:= \arg\min X_{n,m}, \\ (k, l) &:= \arg\min Y_{n,m}. \end{aligned}$$

Case 1: $X_{i,j} < Y_{k,l}$.
If $\mathcal{S} \cup \{(i, j)\} \in \mathcal{X}$,
then $X_{i,j} := \infty$ and $Y_{i,j} := T_{i,j}(t)/\mu + t$,
otherwise $X_{i,j} := T_{i,j}(t)/(\lambda W_{i,j}) + t$.
Case 2: $X_{i,j} > Y_{k,l}$.
Set $\mathcal{S} := \mathcal{S} \setminus \{(k, l)\}$,
$X_{k,l} := T_{k,l}(t)/(\lambda W_{k,l}) + t$ and $Y_{k,l} := \infty$.
Update t:

$$t := t + \min\{X_{i,j}, Y_{k,l}\}.$$

3. Configure the switching fabric according to $\mathcal{S}$.

Elements of X and Y represent times when the corresponding connections are attempted and disconnections are preformed, respectively. An example of how the algorithm evolves over time in a 2×2 switch is shown in Fig. 7. Given the time-encoding computation model, $\mathcal{B}_{(\lambda,\mu,\tau)}$ can be implemented in a distributed fashion. Each reconfiguration of the switching fabric, which either establishes or removes a connection, involves only a single input-output pair – no inter-port communication is required. Therefore, $\mathcal{B}_{(\lambda,\mu,\tau)}$ can be realized by a similar scheme as the one described in Sect. IV-B, with the additional ability to disconnect two matched ports.

Since the elements of X and Y are exponential random variables, the state of the switching fabric can be described by the continuous-time Markov chain $\{\mathcal{S}(t), 0 \le t \le \tau\}$; a state of the switching fabric $\mathcal{I}$ is uniquely defined by the set of edges (matched input-output pairs) in the corresponding bipartite graph. Edges in $\mathcal{I}$, $\mathcal{I} \in \mathcal{X}$, are non-conflicting, i.e., they do not share an input or output node. The state $\emptyset$ denotes

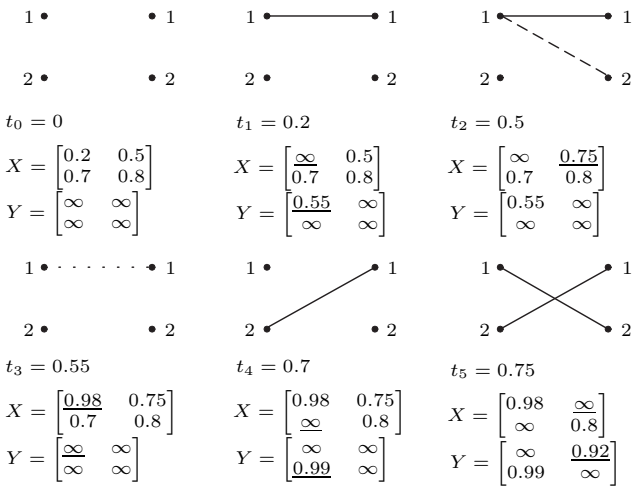

Fig. 7. An example of how the state of the switching fabric evolves over time. The minimum elements in both X and Y at the time instances when changes of state occur are $X_{1,1}$, $X_{1,2}$, $Y_{1,1}$, $X_{2,1}$, $X_{2,2}$. At these time instances, the algorithm changes the matching correspondingly and updates X and Y. The values of X and Y after updates are shown under the corresponding bipartite graphs; the updated elements are underlined. A dashed edge indicates that the connection fails to be established due to a conflict with the current state; a dotted edge indicates that an existing connection is eliminated.

the empty configuration. For every $\mathcal{I} \in \mathcal{X}$, define the following two sets:

$$K_{\mathcal{I}}^{+} = \{\mathcal{J} \in \mathcal{X} : \mathcal{J} = \mathcal{I} \cup \{(k,l)\} \text{ for some } (k,l)\},$$
$$K_{\mathcal{I}}^{-} = \{\mathcal{J} \in \mathcal{X} : \mathcal{I} = \mathcal{J} \cup \{(k,l)\} \text{ for some } (k,l)\}.$$

In short, $K_{\mathcal{I}}^{+}$ is the set of states that $\mathcal{I}$ can augment into with an extra connection, and $K_{\mathcal{I}}^{-}$ is the set of states that $\mathcal{I}$ can reduce into with one removal of a connection. For example, in a 3×3 switch, for $\mathcal{I} = \{(1,1),(2,2)\}$, we have $K_{\mathcal{I}}^{+} = \{\{(1,1),(2,2),(3,3)\}\}$ and $K_{\mathcal{I}}^{-} = \{\{(1,1)\},\{(2,2)\}\}$. The process $\{\mathcal{S}(t), 0 \le t \le \tau\}$ is described by the initial state $\mathcal{S}(0) = \emptyset$ and the rate matrix $[p_{\mathcal{I},\mathcal{J}}]$ with elements

$$p_{\mathcal{I},\mathcal{J}} = \begin{cases} \lambda W_{i,j}, & \mathcal{J} \in K_{\mathcal{I}}^{+} \text{ and } (i,j) \in \mathcal{J} \setminus \mathcal{I}, \\ \mu, & \mathcal{J} \in K_{\mathcal{I}}^{-}, \\ 0, & \text{otherwise.} \end{cases}$$

The state of the switching fabric at the end of a scheduling phase $\mathcal{S}(\tau)$ is fixed for the duration of the cell transfer phase that follows immediately. As λ, $\mu \to \infty$, the distribution of $\mathcal{S}(\tau)$ converges to the stationary distribution π of the process with rate matrix given by $[p_{\mathcal{I},\mathcal{J}}]$. Let $n_{\mathcal{I}}(W)$ denote the number of edges in $\mathcal{I}$ that have a nonzero weight, i.e.,

$$n_{\mathcal{I}}(W) = \sum_{(i,j)\in\mathcal{I}} 1_{\{W_{i,j}>0\}}.$$

We use parameter $\gamma = \lambda/\mu$ to denote the relative rate of connecting/disconnecting ports. Then, the stationary distribution π is given by

$$\pi_{\mathcal{I}} = \pi_{\emptyset}\, \gamma^{n_{\mathcal{I}}(W)} \prod_{(i,j)\in\mathcal{I}} W_{i,j},$$

where

$$\pi_{\emptyset}^{-1} = \sum_{\mathcal{I}\in\mathcal{X}} \gamma^{n_{\mathcal{I}}(W)} \prod_{(i,j)\in\mathcal{I}} W_{i,j}.$$

Indeed, it is straightforward to verify that π is a solution to the set of balance equations, and it satisfies $\sum_{\mathcal{I}\in\mathcal{X}} \pi_{\mathcal{I}} = 1$.

Observe that as $\gamma \to \infty$ and $\mu \to \infty$, algorithm $\mathcal{B}_{(\lambda,\mu,\tau)}$ selects only matchings with the highest size. That is, only matchings from the set $\mathcal{X}^* = \{\mathcal{I} : n_{\mathcal{I}}(W) = \max_{\mathcal{J}\in\mathcal{X}} n_{\mathcal{J}}(W)\}$ are selected with positive probability. In particular, for $\mathcal{I} \in \mathcal{X}^*$,

$$\mathbb{P}[\mathcal{S}(\tau) = \mathcal{I}] \to \prod_{(i,j)\in\mathcal{I}} W_{i,j} / \sum_{\mathcal{J}\in\mathcal{X}^*} \prod_{(i,j)\in\mathcal{J}} W_{i,j},$$

and $\sum_{\mathcal{I}\in\mathcal{X}^*} \pi_{\mathcal{I}} \to 1$, as both $\gamma \to \infty$ and $\mu \to \infty$. Hence, algorithm $\mathcal{B}_{(\lambda,\mu,\tau)}$ implements a weighted version of the MSM algorithm in the limit as $\gamma \to \infty$ and $\mu \to \infty$. For example, when $W_{i,j} = 1_{\{Q_{i,j}>0\}}$, algorithm $\mathcal{B}_{(\lambda,\mu,\tau)}(W)$ selects uniformly among all maximum size matchings. On the other hand, when $W_{i,j}(t) = \exp\{Q_{i,j}(t)\}$ and γ, $\mu \to \infty$, the algorithm configures the switching fabric to state $\mathcal{I}$ with probability

$$\pi_{\mathcal{I}} = \exp\left\{\sum_{(i,j)\in\mathcal{I}} Q_{i,j}(t)\right\} / \sum_{\mathcal{J}\in\mathcal{X}} \exp\left\{\sum_{(i,j)\in\mathcal{J}} Q_{i,j}(t)\right\},$$

i.e., the probability of a matching being scheduled increases exponentially with respect to the number of packets in the matching. In [39] the authors conjecture that a delay-optimal algorithm is the one that selects a max-weight matching among the maximum size matchings (the weight is a logarithmic function). Therefore, by setting $W_{i,j} = Q_{i,j}$, algorithm $\mathcal{B}_{(\lambda,\mu,\tau)}$ approximates (in the limit) this algorithm in [39] since it selects a matching based on $\prod_{(i,j)\in\mathcal{I}} Q_{i,j}$ and $\log \prod_{(i,j)\in\mathcal{I}} Q_{i,j} = \sum_{(i,j)\in\mathcal{I}} \log Q_{i,j}$. Finally, we point out that the structure of stationary distribution π is identical to the stationary distribution of a chain that arises in the analysis of a CSMA scheduling algorithm proposed in [19]. However, this scheduling algorithm is recognized to suffer from the time separation assumption. In particular, the Markov chain can be "trapped" in one stationary state for a long time and hence the biased behavior for each sample. Our approach overcome such bias by refreshing the system state every τ units of time. Given the assumption of a longer transferring phase than scheduling phase, our approach demonstrates the steady state distribution uniformly.

Next we provide some intuition on why reversibility improves switch stability. To this end, consider the example discussed in Sect. IV-C of a 3×3 switch under diagonal traffic with $\alpha = 1/2$. Given that all queues with positive arrival rates are equal and large, algorithm $\mathcal{A}$ selects a matching that serves 2 rather than 3 queues (such as m_3, see (9)) with probability (approximately) $1/3$, i.e., with probability $1/3$ one unit of service is "wasted". This effect results in $\rho^* < 1$ (see Proposition 1). On the other hand, when $\lambda \gg \mu \gg 1/\tau$, algorithm $\mathcal{B}_{(\lambda,\mu,\tau)}$ can configure the switching fabric to a state corresponding to m_3 only at the beginning of the scheduling phase. Given that the switch is in this state, one of the connected pairs will be disconnected (since $\mu \gg 1/\tau$), and,

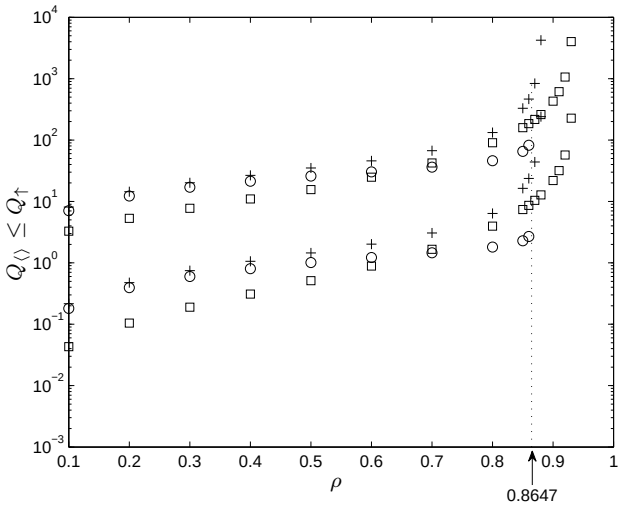

Fig. 8. The expected average virtual output queue length $Q_{\langle\rangle}$ and expected maximum virtual output queue length $Q_{\uparrow}$ for $\mathcal{B}_{(\lambda,\mu,\tau)}(Q)$ (+,□) and an approximation of $\mathcal{A}(Q)$ (○) under diagonal traffic with $\alpha = 1/2$ ($N = 32$). Here $\mathcal{A}(Q)$ is approximated by $\mathcal{B}_{(\lambda,\mu,\tau)}(Q)$ with $(\lambda,\mu,\tau) = (1,0,1)$, and $\mathcal{B}_{(\lambda,\mu,\tau)}(Q)$ uses parameters $(\lambda,\mu,\tau) = (1,10,1)$(+) and $(1,10,10)$(□). The upper bound of the critical ρ for $\mathcal{A}(Q)$ is also shown; note that $\mathcal{B}_{(\lambda,\mu,\tau)}(Q)$ stabilizes the switch under diagonal traffic with values of ρ that exceed this bound.

due to $\lambda \gg \mu$, a feasible pair of ports will be connected. This process continues until 3 pairs are connected, say m_1 or m_2 (see (8)). Once the fabric is in one of these states, it is not configured to a state corresponding to m_3 due to the fact that as soon as one of the pairs is disconnected, the same pair is connected right away ($\lambda \gg \mu$) because it remains the only feasible pair. That is, the limiting algorithm $\mathcal{B}$ selects only among maximum size matchings.

A numerical comparison of $\mathcal{B}_{(\lambda,\mu,\tau)}(Q)$ with parameters $(\lambda,\mu,\tau) = (1,10,1), (1,10,10)$ and an algorithm approximating $\mathcal{A}(Q)$ ($\mathcal{B}_{(\lambda,\mu,\tau)}(Q)$ with $(\lambda,\mu,\tau) = (1,0,1)$) under diagonal traffic is shown in Fig. 8. We point out that $\mathcal{B}_{(\lambda,\mu,\tau)}(Q)$ converges to the stationary distribution fast enough, such that for finite values of (λ,μ,τ) ($(1,10,1)$ in our example), it provides a larger admissible region than $\mathcal{A}(Q)$. Yet, the corresponding queues sizes under $\mathcal{B}_{(\lambda,\mu,\tau)}(Q)$ can be larger than under $\mathcal{A}(Q)$ for moderate values of ρ. This is due to the fact that $\mathcal{B}_{(\lambda,\mu,\tau)}(Q)$ is not efficient when $\lambda \gg \mu$ does not hold, since some feasible input-output pairs might not be connected at the end of the scheduling phase. However, given that $\mathcal{B}_{(\lambda,\mu,\tau)}(Q)$ is determined by the product $\lambda Q_{i,j}$ (see (12)), increasing queue lengths uniformly and multiplicatively is equivalent to increasing λ. Thus, when queue lengths increase, the scheduling performance improves.

Finally, note that the performance of $\mathcal{B}_{(\lambda,\mu,\tau)}(W)$ can be improved by running the algorithm in background during the cell transfer phase, i.e., effectively increasing the value of τ and, thus, allowing for better convergence to the stationary distribution π. Recomputing the schedule for each time slot is equivalent to independently sampling from π in each time slot in the limit as $\gamma, \mu \to \infty$. On the other hand, the algorithm in [19] produces schedules that are strongly correlated in time, and, thus, different behavior. For example, under diagonal traffic with $\alpha = 1/2$ and $Q_{i,j} = q1_{\{\lambda_{i,j}>0\}}$ for large q, $\mathcal{B}_{(\lambda,\mu,\tau)}(W)$ serves each matching with probability $1/2$; on the other hand, if the algorithm is not reset (all ports disconnected) for each time slot, then a matching is selected with probability $1/2$ and served in the following time slots until its weight becomes small relative to the weight of the other matching.

VI. Concluding remarks

This paper introduces a time-encoding model of computation, which enables a set of new architectures and algorithms for packet switches. We developed distributed algorithms for single cross-bar packet switch that achieves performance comparable to that of centralized algorithms. The proposed scheduling algorithms can be applied to scheduling in wireless networks modeled by general interference graphs; in an interference graph, two nodes can not transmit concurrently if they share an edge, see, e.g., [5], [31], [28]. The crossbar switch has a well-structured interference graph (the dual graph of a full bipartite graph) with each node representing an input-output pair. In the case of wireless networks, nodes in the graph represent network nodes and edges typically indicate physical proximity of network noes. Algorithm $\mathcal{A}$ operates on a general graph as follows. Each network node sets possibly multiple timers according to the corresponding weights. A connection is established if the timer expires before any of the interfering pair is scheduled for transmission. The resulting schedule is an approximation of the maximum weight matching on the general graph. Algorithm $\mathcal{B}$ can be applied to general interference graphs analogously by enabling reversibility of scheduling decisions.

Appendix A
Proof of Theorem 1

In order to prove Theorem 1, we introduce the fluid model of a switch. The fluid model is a deterministic, continuous system of equations that parallels (1). Recall that $S_{i,j}(t)$ is the service provided to queue (i,j) in the time slot t. We can rewrite (1) as the following:

$$Q_{i,j}(t) = Q_{i,j}(0) + \sum_{\tau=0}^{t} A_{i,j}(t) - D_{i,j}(t), \qquad (13)$$

where $D_{i,j}(t) = \sum_{\tau=0}^{t} S_{i,j}(\tau)$ is the cumulative number of departures from queue (i,j) up to the time slot t; $D_{i,j}(0) = 0$. For each $t \geq 0$, the evolution of the fluid model is governed by

$$\bar{Q}_{i,j}(t) = \bar{Q}_{i,j}(0) + \lambda_{i,j}t - \bar{D}_{i,j}(t) \geq 0, \qquad (14)$$

where $\bar{Q}_{i,j}(t) = \lim_{r\to\infty} Q(rt)/r$ and $\bar{D}_{i,j}(t) = \lim_{r\to\infty} D(rt)/r$ are the fluid limits of $Q_{i,j}$ and $D_{i,j}$, respectively. A proof of convergence to a unique limit can be found in [8].

Note that $\bar{D}$ depends on the scheduling algorithm. We specify the rate of departure $\dot{\bar{D}}(t)$ at time t in the fluid model under algorithm $\mathcal{A}$ by defining a function ψ; $\dot{\bar{D}}(t)$ denotes

the derivative of $\bar{D}(t)$ at t. For any square matrix X, we use X^{β} to denote a matrix of the same size with elements $X_{i,j}^{\beta}$, $X_{(\lambda)}$ denotes the matrix with elements $X_{i,j}/\lambda_{i,j}$, and $\|X\| = \sum_{i,j=1}^{N} X_{i,j}$. Function ψ represents the expected service rate of the virtual queues under $\mathcal{A}$, and can be recursively defined as follows.

Definition 1: $\psi(X)$ is a matrix with the same dimension as the square matrix X with non-negative elements. If $\|X\| = 0$, then $\psi_{i,j}(X) = 0$ for all i and j. When X is a 1×1 matrix, i.e., a real number, then $\psi(X) = 1_{\{X>0\}}$. For an $N \times N$ matrix X with $\|X\| > 0$, $\psi(X)$ is an $N \times N$ matrix with elements

$$\psi_{i,j}(X) = \frac{X_{i,j}}{\|X\|} + \sum_{k\neq i, l\neq j} \frac{X_{k,l}}{\|X\|} \psi_{i-1_{\{k<i\}}, j-1_{\{l<j\}}}(X_{-(k,l)}),$$

where $X_{-(k,l)}$ is the $(N-1)\times(N-1)$ matrix obtained by deleting the kth row and lth column from X.

For example, if

$$X = \begin{bmatrix} 32 & 10 & 0 \\ 20 & 5 & 1 \\ 25 & 0 & 2 \end{bmatrix},$$

then $\|X\| = 95$ and

$$\begin{aligned}\psi_{2,2}(X) =& \frac{5}{95} + \frac{32}{95}\psi_{1,1}\left(\begin{bmatrix} 5 & 1 \\ 0 & 2 \end{bmatrix}\right) + \frac{25}{95}\psi_{2,1}\left(\begin{bmatrix} 10 & 0 \\ 5 & 1 \end{bmatrix}\right) \\ &+ \frac{2}{95}\psi_{2,2}\left(\begin{bmatrix} 32 & 10 \\ 20 & 5 \end{bmatrix}\right) = \frac{473}{1072}.\end{aligned}$$

Next lemma follows directly from the definition of ψ.

Lemma 1: $\psi(cX) = \psi(X)$ for any constant $c > 0$.

Note that by conditioning on the first timer to expire under $\mathcal{A}(W)$, it is straightforward to obtain $\mathbb{E}S_{i,j}(t) = \psi_{i,j}(W(t))$; recall that $W(t)$ is the weight matrix at time t. From the construction of the fluid model, we have

$$\begin{aligned}\dot{\bar{D}}_{i,j}(t) &= \lim_{\delta\downarrow 0}\lim_{r\to\infty} \frac{1}{r\delta}\left(D_{i,j}(r(t+\delta)) - D_{i,j}(rt)\right)\right) \\ &= \lim_{\delta\downarrow 0}\lim_{r\to\infty} \frac{1}{r\delta} \sum_{\tau=\lfloor rt\rfloor}^{\lfloor r(t+\delta)\rfloor} S_{i,j}(Q^{(\beta)}(\tau)).\end{aligned}$$

For each index pair (i,j), let

$$t_{i,j}^{*} = \frac{1}{r}\arg\max_{\tau=\lfloor rt\rfloor\cdots\lfloor r(t+\delta\rfloor} \psi_{i,j}(Q^{(\beta)}(\tau))$$

be the time index that maximizes function $\psi_{i,j}$ in the specified time interval. We can upper bound $\dot{\bar{D}}_{i,j}$ as follows when $\bar{Q}_{i,j}(t) > 0$:

$$\begin{aligned}\dot{\bar{D}}_{i,j}(t) &\le \lim_{\delta\downarrow 0}\lim_{r\to\infty} \frac{1}{r\delta}\left(\lfloor r(t+\delta)\rfloor - \lfloor rt\rfloor\right)\psi_{i,j}(Q^{(\beta)}(rt_{i,j}^{*})) \\ &= \lim_{\delta\downarrow 0}\lim_{r\to\infty} \psi_{i,j}(r^{-1}Q^{(\beta)})(rt_{i,j}^{*})) \\ &= \lim_{\delta\downarrow 0} \psi_{i,j}(\bar{Q}^{(\beta)}(t_{i,j}^{*})) \\ &= \psi_{i,j}(\bar{Q}^{(\beta)}(t)),\end{aligned}$$

where the second equality is due to Lemma 1 and the third equality follows from the continuity of ψ; it is clear from the definition of $t_{i,j}^{*}$ that $t \le t_{i,j}^{*} \le t+\delta$. Analogously, we can obtain a corresponding lower bound, which combined with the upper bound, yields

$$\dot{\bar{D}}_{i,j}(t) = \psi_{i,j}(\bar{Q}^{(\beta)}(t)), \text{ if } \bar{Q}_{i,j}(t) > 0. \tag{15}$$

Equations (14) and (15) together form the fluid equations of the system operating under algorithm $\mathcal{A}(Q^{(\beta)})$. Any solution $(\bar{D}, \bar{Q})$ to the fluid equations is a fluid model solution under algorithm $\mathcal{A}(Q^{(\beta)})$. A switch operating under algorithm $\mathcal{A}(Q^{(\beta)})$ is rate stable if for every corresponding fluid model solution $(\bar{D}, \bar{Q})$ with $\bar{Q}(0) = 0$, we have $\bar{Q}(t) = 0$ for $t \ge 0$ (see Theorem 3 in [8]). Let $\langle\cdot,\cdot\rangle$ be the element-wise inner product operator of matrices. By Lemma 1 in [8], it is sufficient to show that

$$\frac{d}{dt}\langle \bar{Q}^{(\beta)}(t), \bar{Q}_{(\lambda)}(t)\rangle \le 0 \tag{16}$$

for any $\bar{Q}(t) \neq 0$. The derivative in (16) satisfies

$$\begin{aligned}\frac{d}{dt}\langle \bar{Q}^{(\beta)}(t), \bar{Q}_{(\lambda)}(t)\rangle &= (1+\beta)\langle \bar{Q}^{(\beta)}(t), \dot{\bar{Q}}_{(\lambda)}(t)\rangle \\ &= (1+\beta)\left(\|\bar{Q}^{(\beta)}(t)\| - \langle \bar{Q}^{(\beta)}(t), \dot{\bar{D}}_{(\lambda)}(t)\rangle\right),\end{aligned} \tag{17}$$

where the second equality is due to (14). In the following discussion, index t may be dropped when there is no ambiguity.

(2×2 switch.) For any admissible 2×2 arrival matrix $[\lambda_{i,j}]$, there exists a matrix $[\lambda_{i,j}^{*}]$ such that $\lambda_{i,j}^{*} > \lambda_{i,j}$ and $\lambda_{1,1}^{*} + \lambda_{1,2}^{*} = \lambda_{2,1}^{*} + \lambda_{2,2}^{*} = 1$, since $[\lambda_{i,j}]$ satisfies (4). Definition 1 and (15) yield

$$\dot{\bar{D}}_{i,j}(t) = \begin{cases} M_1\|Q^{(\beta)}\|^{-1}1_{\{Q_{i,i}>0\}}, & i = j, \\ M_2\|Q^{(\beta)}\|^{-1}1_{\{Q_{i,j}>0\}}, & i \neq j, \end{cases}$$

where $M_1 = \bar{Q}_{1,1}^{\beta} + \bar{Q}_{2,2}^{\beta}$ and $M_2 = \bar{Q}_{1,2}^{\beta} + \bar{Q}_{2,1}^{\beta}$, and, thus,

$$\begin{aligned}\langle \bar{Q}^{(\beta)}, \dot{\bar{D}}_{(\lambda)}\rangle &\ge \langle \bar{Q}^{(\beta)}, \dot{\bar{D}}_{(\lambda^{*})}\rangle \\ &= \frac{M_1^2/\lambda_{1,1}^{*} + M_2^2/(1-\lambda_{1,1}^{*})}{\|Q^{(\beta)}\|}.\end{aligned}$$

The preceding inequality and (17) result in

$$\begin{aligned}&\frac{d}{dt}\langle \bar{Q}^{(\beta)}(t), \bar{Q}_{(\lambda)}(t)\rangle \\ &\le \frac{1+\beta}{\|Q^{(\beta)}\|}\left(-\frac{1-\lambda_{1,1}^{*}}{\lambda_{1,1}^{*}}M_1^2 - \frac{\lambda_{1,1}^{*}}{1-\lambda_{1,1}^{*}}M_2^2 + 2M_1M_2\right) \\ &= -\frac{1+\beta}{\|Q^{(\beta)}\|}\left(\sqrt{\frac{1-\lambda_{1,1}^{*}}{\lambda_{1,1}^{*}}}M_1 - \sqrt{\frac{\lambda_{1,1}^{*}}{1-\lambda_{1,1}^{*}}}M_2\right)^2 \le 0.\end{aligned}$$

Hence, the first statement of the theorem follows.

($N\times N$ switch.) Next, we consider a switch of arbitrary size under uniform traffic. The proof is by induction on the number of input/output ports N. In the case of $N = 1$, (16) holds trivially. Now, suppose that (16) holds for a switch with size $(N-1)\times(N-1)$; then (14), (16) and (17) imply

$$\langle \bar{Q}^{(\beta)}, \dot{\bar{D}}_{(\lambda)}\rangle = (N-1)\sum_{i,j=1}^{N-1} \bar{Q}_{i,j}^{\beta}\psi_{i,j}(\bar{Q}^{(\beta)}) \ge \|\bar{Q}^{(\beta)}\|, \tag{18}$$

where $\bar{Q}$ is a $(N-1)\times(N-1)$ matrix. For a switch of size $N\times N$, combining (14), Definition 1 and $\lambda_{i,j}=1/N$ yields

$$\begin{aligned}
&\langle \bar{Q}^{(\beta)}, \dot{\bar{D}}_{(\lambda)}\rangle \\
&= N\sum_{i,j=1}^{N} \bar{Q}_{i,j}^{\beta}\left(\frac{\bar{Q}_{i,j}^{\beta}}{\|\bar{Q}^{(\beta)}\|}+\sum_{k\neq i,l\neq j}\frac{\bar{Q}_{k,l}^{(\beta)}}{\|\bar{Q}^{(\beta)}\|}\psi_{i,j}(\bar{Q}_{-(k,l)}^{(\beta)})\right)\\
&= \frac{N}{\|\bar{Q}^{(\beta)}\|}\left(\|\bar{Q}^{(2\beta)}\|+\sum_{k,l=1}^{N}\bar{Q}_{k,l}^{\beta}\sum_{i\neq k,j\neq l}\bar{Q}_{i,j}^{\beta}\psi_{i,j}(\bar{Q}_{-(k,l)}^{(\beta)})\right)\\
&\geq \frac{N}{\|\bar{Q}^{(\beta)}\|}\left(\|\bar{Q}^{(2\beta)}\|+\frac{1}{N-1}\sum_{k,l=1}^{N}\bar{Q}_{k,l}^{\beta}\|\bar{Q}_{-(k,l)}^{(\beta)}\|\right), \quad (19)
\end{aligned}$$

where the inequality follows from applying the inductive hypothesis (18) to matrix $\bar{Q}_{-(k,l)}^{(\beta)}$. Next, let $q_{i,j}=\bar{Q}_{i,j}^{\beta}/\|\bar{Q}^{(\beta)}\|$ and rewrite the right-hand side of (19) as

$$\begin{aligned}
&N\|\bar{Q}^{(\beta)}\|\sum_{k,l=1}^{N} q_{k,l}\left(q_{k,l}+\frac{1}{N-1}\sum_{i\neq k,j\neq l}q_{i,j}\right)\\
&\geq N\|\bar{Q}^{(\beta)}\|\inf_{\{q_{k,l}\}}\left\{\sum_{k,l=1}^{N} q_{k,l}\left(q_{k,l}+\frac{1}{N-1}\sum_{i\neq k,j\neq l}q_{i,j}\right)\right\},
\end{aligned}$$

where the infimum is over all $q_{k,l}\in[0,1]$ such that $\sum_{k,l}q_{k,l}=1$. The infimum is attained when $q_{k,l}=1/N^2$ for all k, l, resulting in

$$\langle \bar{Q}^{(\beta)}, \dot{\bar{D}}_{(\lambda)}\rangle \geq \|\bar{Q}^{(\beta)}\|.$$

The preceding inequality, (17) and (18) imply the second statement of the theorem.

Appendix B
Proof of Proposition 1

Consider the fluid model described in Appendix A:

$$\begin{cases}\bar{Q}_{i,j}(t)=\bar{Q}_{i,j}(0)+\lambda_{i,j}t-\bar{D}_{i,j}(t),\\ \dot{\bar{D}}_{i,j}(t)=\psi_{i,j}(\bar{Q}^{(\beta)}(t)),\ \text{if } \bar{Q}_{i,j}(t)>0.\end{cases} \quad (20)$$

We show that $\bar{Q}_{i,j}(t)\to\infty$ as $t\to\infty$ for $j=i$ or $(i \bmod N)+1$, under the diagonal traffic with $\alpha=1/2$ and some $\rho\in(0,1)$, and the following initial condition:

$$\bar{Q}_{i,j}(0)=\begin{cases}q, & j=i \text{ or } (i \bmod N)+1,\\ 0, & \text{otherwise,}\end{cases}$$

where $q>0$, i.e., $\bar{Q}_{i,j}(t)=0$ for all $t\geq 0$ and i, j such that $\lambda_{i,j}=0$. Given this initial condition, due to a symmetry in ψ and the arrival matrix, as well as Lemma 1, (20) reduces to

$$\bar{Q}_{i,j}(t)=\begin{cases}q+\rho t/2-\sigma t, & j=i \text{ or } (i \bmod N)+1,\\ 0, & \text{otherwise.}\end{cases}$$

for all $t\geq 0$ when $\rho/2>\sigma$ with σ being the service rate:

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T}S_{i,j}(t)=\begin{cases}\sigma, & j=i \text{ or } (i \bmod N)+1,\\ 0, & \text{otherwise.}\end{cases}$$

That is, for any $\rho>2\sigma$, the algorithm is not rate stable.

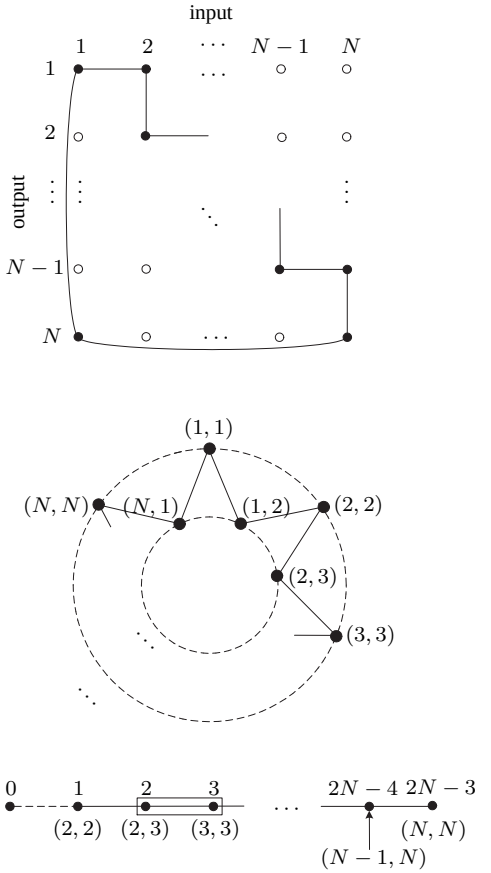

Fig. 9. The interference graph for an $N\times N$ switch under diagonal traffic is shown on the top, where nodes correspond to queues with positive occupancies and arrival rates; edges connect queues with a common input/output port. On the top, the interference graph is drawn according to the relative location of the nodes in the traffic matrix; in the middle the graph is drawn on two circles, each circle refers to a matching of size N. Conditioning on the event that queue $(1,1)$ is scheduled first, the remaining interference graph is linear as shown on the bottom of the figure. Here an additional fictitious node 0 is added to the left end of the line in order to draw an equivalency with a dimer packing problem considered in [32]. A dimer is placed on nodes 2 and 3, indicating that queue $(3,3)$ is scheduled. As a result, queues $(2,3)$ and $(3,4)$ can not be scheduled in the same time slot, since dimers are not allowed to overlap.

In order to determine the service (departure) rate σ, we construct the interference graph of the switch under the diagonal traffic (see the top of Fig. 9). Virtual queue (i,j) with positive arrival rate is represented as the node (i,j); the total number of nodes in the interference graph is $2N$. In this graph, two nodes are connected by an edge if they can not be scheduled concurrently. The graph under diagonal traffic is a $2N$-cycle, exhibiting the symmetry of the queues with positive arrival rates. Due to this symmetry, the service rate σ satisfies $\sigma=s_N/2N$, where s_N is the expected number of scheduled queues in a single time slot. Without loss of generality (due to symmetry), we assume that queue $(1,1)$ is scheduled first, then it follows that queues $(1,2)$ and $(N,1)$ can not be scheduled in the same time slot. Thus, the remaining interference graph is linear with $2N-3$ nodes, as shown in the bottom of Fig. 9. We argue that determining the expected number of

served queues on a linear interference graph is equivalent to computing the expected number of vacancies on a line packed with randomly placed dimers (non overlapped adjacent pairs of nodes). For convenience, in the following discussion, we relabel the nodes in the interference graph by natural numbers. With an additional fictitious node 0 on the left end of the line interference graph, placing a dimer at node pair $\{i-1, i\}$ is equivalent to an expiration of a timer corresponding to node i. In particular, when a dimer is placed at a node pair $\{i-1, i\}$, then a dimer can not be placed at node pairs $\{i-2, i-1\}$ and $\{i, i+1\}$ since dimers are not allowed to overlap. This is equivalent to preventing queues corresponding to nodes $i-1$ and $i+1$ from being scheduled when the queue corresponding to node i is already scheduled. Hence, if x_k is the expected number of vacancies on a line graph with k nodes (including the node 0), then the following relationship must hold:

$$(2N-3)+1 = 2(s_N - 1) + x_{(2N-3)+1}.$$

The value of x_k was obtained by Page [32]:

$$x_k = (k+2)\left(\sum_{r=0}^{k+1}\frac{(-2)^r}{r!} + \frac{1}{2}\frac{(-2)^{k+2}}{(k+2)!}\right),$$

and, therefore, we have

$$\begin{aligned} s_N &= N - \frac{1}{2}x_{2N-2} \\ &= N\left(1 - \left[\sum_{r=0}^{2N-1}\frac{(-2)^r}{r!} + \frac{1}{2}\frac{(-2)^{2N}}{(2N)!}\right]\right). \end{aligned}$$

Recall that the critical relative load ρ^* can be at most s_N/N. For example, the preceding expression yields $s_{32} \approx 27.6693\ldots$, indicating that $\rho^* < 0.8647\ldots$ for a 32×32 switch (see the top of Fig. 6). In addition, it also follows that $\rho^* < 1 - e^{-2}$ in the limit as $N \to \infty$.

APPENDIX C
A PROCEDURE FOR ESTIMATING ρ^* FOR $\alpha \in (0,1)$

In order to upper bound the maximum relative load under diagonal traffic, we consider the fluid model analyzed in Appendix B. In particular, we focus on the fluid model (20) with the following initial condition:

$$\bar{Q}_{i,j}(0) = \begin{cases} \kappa q, & j = i, \\ (1-\kappa)q, & j = (i \bmod N) + 1, \\ 0, & \text{otherwise}, \end{cases} \tag{21}$$

where $\kappa \in [0,1]$ is such that

$$\frac{\psi_{i,i}(\bar{Q}^{(\beta)}(0))}{\psi_{i,(i \bmod N)+1}(\bar{Q}^{(\beta)}(0))} = \frac{\alpha}{1-\alpha}. \tag{22}$$

Note that such κ exists for any $\alpha \in [0,1]$ due to continuity and symmetry of ψ. Due to symmetry of the arrival matrix, under (21) and (22), (20) renders for all $t \geq 0$, when $\rho > \sigma$,

$$\bar{Q}_{i,j}(t) = \begin{cases} \kappa q + \alpha(\rho-\sigma)t, & j = i, \\ (1-\kappa)q + (1-\alpha)(\rho-\sigma)t, & j = (i \bmod N) + 1, \\ 0, & \text{otherwise}, \end{cases}$$

where σ is the total service rate of queues (i,i) and $(i,(i \bmod N)+1)$. In this case, the algorithm is not rate stable.

The value of κ that yields (22) can be evaluated numerically since ψ is defined recursively in Definition 1. Once this value is obtained, one can obtain $\sigma = \psi_{1,1}(\bar{Q}(0)) + \psi_{1,2}(\bar{Q}(0))$ and an upper bound $\rho^* < \sigma$ follows. For example, when $N = 32$ and $\alpha = 2/3$, we have $\kappa = 0.6295\ldots$, $\psi_{1,1}(\bar{Q}(0)) = 0.5829\ldots$, $\psi_{1,2}(\bar{Q}(0)) = 0.2915\ldots$ and $\sigma = 0.8744\ldots$.

REFERENCES

[1] L. Adleman. Molecular computation of solutions to combinatorial problem. *Science*, 266:1021–1024, 1994.

[2] M. Andrews, K. Kumaran, K. Ramanan, A. Stolyar, R. Vijayakumar, and Ph. Whiting. Scheduling in a queuing system with asynchronously varying service rates. *Probab. Engin. Inform. Sci.*, 18:191–217, 2004.

[3] B. Ata and W. Lin. Heavy traffic analysis of maximum pressure policies for stochastic processing networks with multiple bottlenecks. *Queueing Syst. Theory Appl.*, 59(3-4):191-235, 2008.

[4] M. Bayati, B. Prabhakar, D. Shah, and M. Sharma. Iterative scheduling algorithms. In *Proc. IEEE Infocom*, Anchorage, AK, May 2007.

[5] P. Chaporkar, K. Kar, and S. Sarkar. Throughput guarantees through maximal scheduling in wireless networks. In *Proc. of Allerton Conf. on Comm., Control, and Comput.*, Monticello, IL, September 2005.

[6] J. Dai. A fluid model criterion for instability of multiclass queueing networks. *Ann. Appl. Probab.*, 6(3):751–757, 1996.

[7] J. Dai and W. Lin. Asymptotic optimality of maximum pressure policies in stochastic processing networks. *Ann. Appl. Probab.*, 18:2239–2299, 2008.

[8] J. Dai and B. Prabhakar. The throughput of data switches with and without speedup. In *Proc. IEEE Infocom*, Tel Aviv, Israel, March 2000.

[9] S. Deb, D. Shah, and S. Shakkottai. Fast matching algorithms for repetetive optimization: An application to switch scheduling. In *Proc. CISS*, Princeton, NJ, March 2006.

[10] A. Dimakis and J Walrand. Sufficient conditions for stability of longest queue first scheduling: Second order properties using using fluid limits. *Adv. Appl. Probab.*, 38(2):505–521, 2006.

[11] E.A. Dinic. Algorithm for solution of a problem of maximum flow in a network with power estimation. *Soviet Math. Dokl.*, 11:1277–1280, 1970.

[12] H. Duan, J. Lockwood, and S.M. Kang. Matrix unit cell scheduler (MUCS) for input-buffered ATM switches. *IEEE Commun. Letters*, 2(1):20–23, 1998.

[13] N.G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *IEEE/ACM Trans. Networking*, 9(3):280–292, 2001.

[14] A. Eryilmaz, A. Ozdaglar, and E. Modiano. Polynomial complexity algorithms for full utilization of multi-hop wireless networks. In *Proc. IEEE Infocom*, Anchorage, AK, May 2007.

[15] H.N. Gablow and R.E. Tarjan. Faster scaling algorithms for general graph matching problems. *J. ACM*, 38(4):815–853, 1991.

[16] P. Giaccone, B. Prabhakar, and D. Shah. Randomized scheduling algorithms for high-aggregate bandwidth switches. *IEEE J. Select. Areas Commun.*, 21(4):546–559, 2003.

[17] P. Giaccone, D. Shah, and B. Prabhakar. An implementable parallel scheduler for input-queued switches. *IEEE Micro*, January-February 2002.

[18] J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM J. Comput.*, 2:225–231, 1973.

[19] L. Jiang and J. Walrand. A distributed CSMA algorithm for throughput and utility maximization in wireless networks. In *Proc. of Allerton Conf. on Comm., Control, and Comput.*, Monticello, IL, September 2008.

[20] I. Keslassy, C.-S. Chang, N. McKeown, and D.-S. Lee. Optimal load-balancing. In *Proc. IEEE Infocom*, Miami, FL, March 2005.

[21] I. Keslassy, S.-T. Chuang, and N. McKeown. A load-balanced switch with an arbitrary number of lines. In *Proc. IEEE Infocom*, Hong Kong, March 2004.

[22] I. Keslassy, R. Zhang-Shen, and N. McKeown. Maximum size matching is unstable for any packet switch. *IEEE Commun. Letters*, 7(10):496–498, 2003.

[23] M. Ajmone Marsan, A. Bianco, E. Leonardi, and L. Milia. RPA: A flexible scheduling algorithm for input buffered switches. *IEEE Trans. Commun.*, 47(12):1921–1933, 1999.

[24] M.G. Ajmone Marsan, P. Giaccone, E. Leonardi, and F. Neri. On the stability of local scheduling policies in networks of packet switches with input queues. *IEEE J. Select. Areas Commun.*, 21(4):642–655, 2003.
[25] N. McKeown. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Trans. Networking*, 7(2):188–201, 1999.
[26] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. *IEEE Trans. Comm.*, 47(8):1260–1267, 1999.
[27] P. Momčilović. A distributed switch scheduling algorithm. (In *Proc. of Performance*, Cologne, Germany, October 2007.) *Perform. Eval.*, 64(9-12):1053–1061, 2007.
[28] P. Mani and D. Petr. Clique number vs. chromatic number in wireless interference graphs: Simulation results. *IEEE Commun. Letters*, 11(7):592–594, 2007.
[29] E. Modiano, D. Shah, and G. Zussman. Maximizing throughput in wireless networks via gossiping. In *Proc. ACM Sigmetrics*, Saint Malo, France, June 2006.
[30] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, UK, 2000.
[31] P. Ng, D. Edwards and S. Liew. Colouring link-directional interference graphs in wireless ad hoc networks. In *Proc. IEEE Globecom*, Washington, DC, November 2007.
[32] E. Page. The distribution of vacancies on a line. *J. Royal Stat. Soc. B*, 21(2):364–374, 1959.
[33] G. Paun. *Membrane computing*. Springer, Berlin, 2002.
[34] S. Sarkar and K. Kar. Achieving 2/3 throughput approximations with sequential maximal scheduling under primary interference constraints. In *Proc. of Allerton Conf. on Comm., Control, and Comput.*, Monticello, IL, September 2006.
[35] D. Shah. *Randomization and heavy traffic: New approaches for switch algorithms*. PhD thesis, Stanford University, 2004.
[36] D. Shah and M. Kopikare Delay bounds for the approximate maximum weight matching algorithm for input queued switches. In *Proc. IEEE Infocom*, New York, NY, June 2002.
[37] G. Sharma, N. Shroff, and R. Mazumdar. Joint jongestion control and distributed scheduling for throughput guarantees in wireless networks. In *Proc. IEEE Infocom*, Anchorage, AK, May 2007.
[38] A. Stolyar. MaxWeight scheduling in a generalized switch: State space collapse and workload minimization in heavy traffic. *Ann. Appl. Probab.*, 14(1):1–53, 2004.
[39] D. Shah and D. Wischik. Optimal scheuling algorithms for input-queued switches. In *Proc. IEEE Infocom*, Barcelona, Spain, 2006.
[40] L. Tassiulas. Adaptive back-pressure congestion control based on local information. *IEEE Trans. Automat. Control*, 40(2):236–250, 1995.
[41] L. Tassiulas. Linear complexity algorithms for maximum throughput in radio networks and input queued switches. In *Proc. IEEE Infocom*, San Francisco, CA, April 1998.
[42] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling for maximum throughput in multihop radio networks. *IEEE Trans. Automat. Control*, 37(12):1936–1949, 1992.
[43] K. Winick. Personal communication.