A-MWM: A Low Complexity Packet-Mode Scheduling Algorithm

Shunyuan Ye, Yanming Shen, Shivendra Panwar

Abstract

In a packet switching system, arriving packets have variable lengths. They are segmented into fixed size cells before being switched. In packet-mode scheduling, cells from a packet should be delivered contiguously. Even though fixed-size cell scheduling has been well studied, packet-mode scheduling has more advantages: it has a lower implementation cost and higher throughput. In this paper, we first propose a centralized algorithm for an input-queued switch with a time complexity of $O(N)$. By making some simple modifications, the complexity can be reduced to $O(1)$. We prove that the algorithm can stabilize the system for admissible i.i.d arrivals with any finite packet size distribution. We then extend the idea and show that it can be implemented in a buffered crossbar switch in a distributed manner. Our simulation results show that it can provide good delay performance for different traffic patterns with different packet size distributions.

I. Introduction

The seminal work by Tassiulas and Ephremides [1] initiated the study of scheduling algorithms in packet switching systems and wireless networks. *Maximum weight matching* (**MWM**), which was proposed in [1], can stabilize the system whenever the arrival traffic is admissible. But it is not practical to implement due to two drawbacks: i) it is of very high computation complexity; ii) the algorithm is centralized. For *input-queued* (**IQ**) switches, the computation complexity of MWM is $O(N^3)$, where N is the size of a switch.

In the past decade, there have been a lot of research effort trying to propose practical algorithms that either are of low complexity or can be implemented in a distributed way. Most of these algorithms assume that packets are segmented into fixed-size cells before scheduling, and after traversing the switching fabric, cells are reassembled back into original packets at the output ports. Cells from a packet are scheduled independently. A packet can leave the system only when all of its cells are received. These algorithms are referred to as *cell-mode scheduling*.

Marsan *et al.* [2], on the other hand, proposed to switch variable-size packets directly, which is referred to as *packet-mode scheduling*. In packet-mode scheduling, the input ports have to deliver all the cells from the segmentation of a packet contiguously. Thus, the reassembly of packets at the output ports is much easier and requires less memory and complexity. They showed that for some packet size distributions, packet-mode scheduling can have a better delay performance, compared to the cell-mode scheduling. An algorithm (PI-MWM) is proposed, which can stabilize the system for any admissible Bernoulli arrivals. However, the algorithm is a modification of MWM. Therefore, it is still centralized and of high computation complexity. Ganjali *et al.* [3] show that PI-MWM is stable for any form of re-generative admissible traffic, rather than only Bernoulli i.i.d.

In this paper, we revisit the packet-mode scheduling algorithm. We are interested in this problem due to the fact that for some applications, packet-mode scheduling is more attractive. For example, in optical networks, it is difficult to segment a packet. Also, it eliminates the need to implement a reassembly stage at the output ports, as shown in Fig. 1. The objective of this paper is to design a packet-mode scheduling algorithm with low computation complexity and easy hardware implementation. We first propose a centralized scheduling algorithm *Approximate MWM (A-MWM)* for an input-queued switch, which has a complexity of $O(N)$. The complexity can be reduced to $O(1)$ by making some simple modifications. We prove that the algorithm can stabilize the system for any admissible i.i.d arrival process. The idea is then extended to a buffered crossbar switch, where the algorithm proposed can be easily implemented in a distributed manner. Our simulation results show that the algorithm can provide good delay performance for different arrival traffic patterns. Note that when all packets have unit size, A-MWM also is a low-complexity cell-mode scheduling algorithm.

The paper is organized as follows. In Section II, we present a centralized, low-complexity scheduling algorithm in an input-queued switch and present the simulation results in Section III. The idea is extended to a buffered crossbar switch in Section IV. We then present some simulation results for the buffered crossbar switch in Section V. Section VI concludes the paper.

II. Input-queued Switches

A. System Model

An $N \times N$ input-queued switch is shown in Fig. 1. The packet arrival process at an input is characterized by a two-state slotted time ON-OFF model. Packet lengths are expressed in slot times (or cells). When it

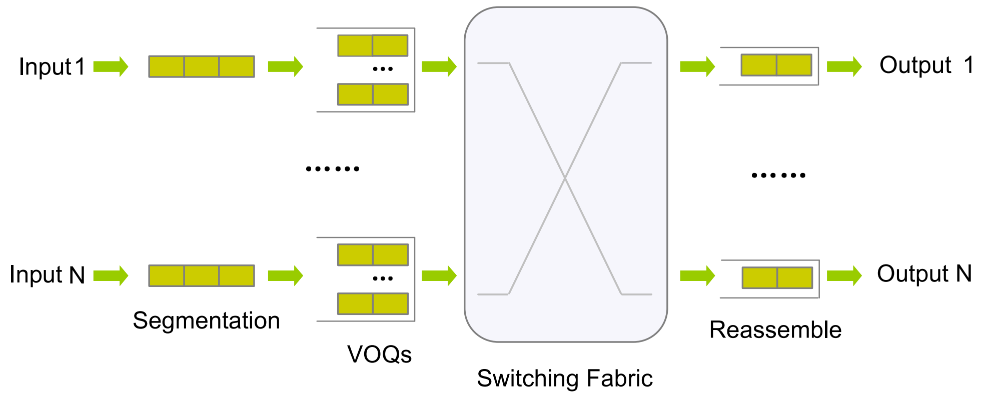

Fig. 1: Packet-Mode Input-Queued Switch

is in the ON state, a packet is being received. The number of time slots spent in the ON state is the length of the packet. No packets arrive in OFF state. The number of slots spent in OFF state is geometrically distributed.

The packet length can have any distribution as long as the maximum packet length is finite. There are *virtual output queues* (VOQ) at the inputs to prevent *head-of-line blocking*. Each input maintains N VOQs, one for each output. Let VOQ_{ij} represent the VOQ at input i for output j. Let $Q_{ij}(n)$ denote the total queued packet length expressed in the number of cells in VOQ_{ij} at time n, and $\mathbf{Q} = \{Q_{ij}\}$. Let (i, j) represent the crosspoint between input i and output j. Note that a VOQ corresponds to a crosspoint.

A schedule can be represented by a matrix $\mathbf{S} \in \{0, 1\}^{N\times N}$. $S_{ij} = 1$ if VOQ_{ij} is in the schedule; otherwise, $S_{ij} = 0$. With some abuse of notation, we also use $\mathbf{S}$ to represent a set, and write $(i, j) \in \mathbf{S}$ if $S_{ij} = 1$.

Definition 1: A *feasible schedule* $\mathbf{S}(n)$ is an $N \times N$ matrix, where $S_{ij}(n) \in \{0, 1\}$, and $\sum_i S_{ij}(n) \leq 1$, $\sum_j S_{ij}(n) \leq 1$.

Note that a feasible schedule $\mathbf{S}$ has the property that if $S_{ij} = 1$, then $\forall i' \neq i$, $S_{i'j} = 0$ and $\forall j' \neq j$, $S_{ij'} = 0$. We define these crosspoints as its *neighbors*.

Definition 2: For a crosspoint (i, j), its neighbors are defined as:

$$\mathcal{N}(i, j) = \{(i', j) \text{ or } (i, j') \mid \forall i' \neq i,\ \forall j' \neq j\} \tag{1}$$

So for a feasible schedule $\mathbf{S}$, if $(i, j) \in \mathbf{S}$, then $\forall (k, l) \in \mathcal{N}(i, j)$, $(k, l) \notin \mathbf{S}$. Let $\mathcal{S}$ represent the set of all feasible schedules.

Let λ_{ij} represent the arrival rate of traffic between input i and output j. We assume that the arrival is a Bernoulli process.

Definition 3: An arrival process is said to be *admissible* if it satisfies:

$$\sum_j \lambda_{ij} < 1, \text{ and } \sum_i \lambda_{ij} < 1. \tag{2}$$

Let $||\mathbf{Q}||$ represent the norm of matrix **Q**: $||\mathbf{Q}|| = \left(\sum_{i,j} Q_{ij}^2\right)^{1/2}$. The stability of a system is defined as:

Definition 4: A system of queues is said to be *stable* if:

$$\lim_{n\to\infty} \sup E||\mathbf{Q}(n)|| < \infty. \tag{3}$$

B. Packet-Mode Scheduling Algorithm

Basic Scheduling Algorithm of A-MWM

```
∘ $\forall\ (i,j) \notin \mathbf{H}(n)$:
    (a) $S_{ij}(n) = S_{ij}(n-1)$.
∘ For $(i,j) \in \mathbf{H}(n)$:
    - If $(i,j) \in \mathbf{S}(n-1)$ and the packet transmission between input $i$ and output $j$ has not
      been completed:
        (b) $S_{ij}(n) = S_{ij}(n-1) = 1$.
    - Else, if $(i,j) \in \mathbf{S}(n-1)$ and the packet transmission has been finished:
        (c) $S_{ij}(n) = 1$ with probability $p_{ij}$;
        (d) $S_{ij}(n) = 0$ with probability $\overline{p}_{ij} = 1 - p_{ij}$.
    - Else, if $(i,j) \notin \mathbf{S}(n-1)$, and $\forall(k,l) \in \mathcal{N}(i,j)$, $S_{kl}(n-1) = 0$, then:
        (e) $S_{ij}(n) = 1$ with probability $p_{ij}$;
        (f) $S_{ij}(n) = 0$ with probability $\overline{p}_{ij} = 1 - p_{ij}$.
    - Else, $(i,j) \notin \mathbf{S}(n-1)$, and $\exists(k,l) \in \mathcal{N}(i,j)$ such that $S_{kl}(n-1) = 1$:
        (g) $S_{ij}(n) = 0$.
```

In our algorithm, the system has to keep track of the schedule of previous time slot $\mathbf{S}(n-1)$. At the beginning of time n, generate a feasible schedule $\mathbf{H}(n)$ by using a Hamiltonian walk [4]. For an input-queued switch, there are $N!$ distinct matchings. A Hamiltonian walk $\mathbf{H}(n)$ visits each of the $N!$ distinct matchings exactly once during times $n = 1, 2, ..., N!$. For $n > N!$, $\mathbf{H}(n) = \mathbf{H}(n \bmod N!)$. A Hamiltonian walk can be generated with a simple algorithm with a time complexity of $O(1)$ [5].

After $\mathbf{H}(n)$ is generated, the schedule $\mathbf{S}(n)$ is decided following the algorithm above. The probability p_{ij} is a concave function (to be specified later) of the queue size Q_{ij}. Note that in our algorithm, $S_{ij}(n)$ can change only when the VOQ_{ij} is selected by $\mathbf{H}(n)$.

C. Stationary Distribution

Define the state:

$$\mathbf{Y} = (\mathbf{S}, \mathbf{L}, \mathbf{R}), \tag{4}$$

where $\mathbf{L} = \{L_{ij}\}$, L_{ij} is the size of the packet being transmitted between input i and output j. $\mathbf{R} = \{R_{ij}\}$, and R_{ij} is the remaining time needed to complete the transmission of the packet between input i and output j. Let $\mathcal{Y}$ represent the set of all possible states.

Define the active VOQs whose remaining transmission time is larger than 1 at the beginning of time $n-1$ as:

$$\mathbf{B}(n-1) = \{(i,j)|S_{ij}(n-1) = 1,\ R_{ij}(n-1) > 1\}. \tag{5}$$

On the other hand, active VOQs, whose remaining transmission time was 1, already finished their packet transmissions at time $n-1$. According to the A-MWM scheduling algorithm, they are possible to change their schedules.

According to the scheduling algorithm, inactive VOQs can join the schedule only when none of its neighbors was in the schedule of the previous time slot. Therefore, at time n, all the neighbors of **S**(n-1) are blocked, and will remain inactive. So all the VOQs in $\mathbf{B} \cup \mathcal{N}(\mathbf{S})$ will keep their schedules of the previous time slot. Only VOQs in $\overline{\mathbf{B} \cup \mathcal{N}(\mathbf{S})}$ can change their schedules if they are picked by the Hamiltonian walk schedule $\mathbf{H}(n)$.

Lemma 1: A state $\mathbf{Y} = (\mathbf{S}, \mathbf{L}, \mathbf{R})$ can transit to a state $\mathbf{Y}' = (\mathbf{S}', \mathbf{L}', \mathbf{R}')$ if and only if $\mathbf{Y}'$ satisfies the conditions below:

1) If $(i,j) \in \mathbf{B}$, $S'_{ij} = S_{ij} = 1$, $L'_{ij} = L_{ij}$, and $R'_{ij} = R_{ij} - 1$.
2) $\mathbf{S} \cup \mathbf{S}' \in \mathcal{S}$.

Proof: (Necessity) In packet-mode scheduling, once start a packet transmission, the input and output have to keep serving this packet until all the cells from the packet have been delivered to the output. Therefore, for those active VOQs whose remaining time for the current packet transmission was larger than 1, they should stay in the schedule, and the remaining time will be reduced by one since one cell was transmitted in previous time slot. So condition 1) is necessary.

Following the definition, a schedule $\mathbf{S}$ is feasible if for any $(i,j) \in \mathbf{S}$, $\forall (k,l) \in \mathcal{N}(i,j)$, $(k,l) \notin \mathbf{S}$. $\mathbf{S} \cup \mathbf{S}' = \mathbf{S} \cup (\overline{\mathbf{S}} \cap \mathbf{S}')$. $\mathbf{S}$ and $\overline{\mathbf{S}} \cap \mathbf{S}'$ are disjoint. So for any $(i,j) \in \mathbf{S} \cup \mathbf{S}'$, (i,j) belongs to one of these two sets: $\mathbf{S}$ or $\overline{\mathbf{S}} \cap \mathbf{S}'$.

If $(i,j) \in \mathbf{S}$, $\forall (k,l) \in \mathcal{N}(i,j)$, $(k,l) \notin \mathbf{S}$ since $\mathbf{S}$ itself is a feasible schedule. According the scheduling algorithm, VOQs can join the schedule only when none of its neighbors was in the schedule of the previous time slot. Thus, $(k,l) \notin \mathbf{S}'$, since there is a neighbor (i,j) such that $S_{ij} = 1$. So, $\forall (k,l) \in \mathcal{N}(i,j)$, $(k,l) \notin \mathbf{S}$ and $(k,l) \notin \mathbf{S}'$. Thus, $(k,l) \notin \mathbf{S} \cup \mathbf{S}'$.

If $(i,j) \in \overline{\mathbf{S}} \cap \mathbf{S}'$, $\forall (k,l) \in \mathcal{N}(i,j)$, $(k,l) \notin \mathbf{S}'$ since $\mathbf{S}'$ is also a feasible schedule. (k,l) can not be in $\mathbf{S}$ as (i,j) changes its schedule decision from 0 to 1. if (k,l) is in $\mathbf{S}$, (i,j) can not join the schedule according the algorithm. So, $\forall (k,l) \in \mathcal{N}(i,j)$, $(k,l) \notin \mathbf{S}'$ and $(k,l) \notin \mathbf{S}$. Thus, $(k,l) \notin \mathbf{S} \cup \mathbf{S}'$.

We already proved that if a state $\mathbf{Y} = (\mathbf{S}, \mathbf{L}, \mathbf{R})$ can transit to a state $\mathbf{Y}' = (\mathbf{S}', \mathbf{L}', \mathbf{R}')$, then for any $(i,j) \in \mathbf{S} \cup \mathbf{S}'$, $\forall (k,l) \in \mathcal{N}(i,j)$, $(k,l) \notin \mathbf{S} \cup \mathbf{S}'$. Therefore, $\mathbf{S} \cup \mathbf{S}'$ is feasible.

(Sufficiency) If $\mathbf{S} \cup \mathbf{S}' \in \mathcal{S}$, then $(\mathbf{S} \cap \overline{\mathbf{S}'}) \cup (\overline{\mathbf{S}} \cap \mathbf{S}') \in \mathcal{S}$. The Hamiltonian walk visits all feasible schedules. Thus there exists at least one $\mathbf{H}(n)$ such that $(\mathbf{S} \cap \overline{\mathbf{S}'}) \cup (\overline{\mathbf{S}} \cap \mathbf{S}') \in \mathbf{H}(n)$. When these $\mathbf{H}(n)$ are selected and $\mathbf{Y}'$ also satisfies condition 1), $\mathbf{Y}$ can transit to $\mathbf{Y}'$ with a probability larger than zero which we will give below. **QED** ■

Lemma 2: If a state $\mathbf{Y} = (\mathbf{S}, \mathbf{L}, \mathbf{R})$ can transit to a state $\mathbf{Y}' = (\mathbf{S}', \mathbf{L}', \mathbf{R}')$, the transition probability then is:

$$\begin{aligned} P(\mathbf{Y}, \mathbf{Y}') \quad &= \sum_{\mathbf{H}:\mathbf{S} \triangle \mathbf{S}' \in \mathbf{H}} a(\mathbf{H}) \prod_{(i,j) \in \mathbf{S} \cap \overline{\mathbf{S}}'} \overline{p}_{ij} \prod_{(k,l) \in \overline{\mathbf{S}} \cap \mathbf{S}'} p_{kl} \times P_{kl}(L'_{kl}) \\ &\bullet \prod_{(u,v) \in (\mathbf{S} \cap \overline{\mathbf{B}}) \cap (\mathbf{S}' \cap \overline{\mathbf{B}}) \cap \mathbf{H}} p_{uv} \times P_{uv}(L'_{uv}) \prod_{(x,y) \in \mathbf{H} \cap \overline{\mathbf{S} \cup \mathbf{S}'} \cap \overline{\mathcal{N}(\mathbf{S} \cup \mathbf{S}')}} \overline{p}_{xy}, \end{aligned} \tag{6}$$

where $P_{ij}(L'_{ij})$ is the probability that the head-of-line packet in VOQ_{ij} has a size of L'_{ij}. We assume that the packet length distribution for each VOQ is fixed, namely, $P_{ij}(L'_{ij})$ is a constant. $a(\mathbf{H})$ is the probability that $\mathbf{H}$ is selected (which is $1/N!$), and $\mathbf{S} \triangle \mathbf{S}'$ = $(\mathbf{S} \cap \overline{\mathbf{S}'}) \cup (\overline{\mathbf{S}} \cap \mathbf{S}')$.

Proof: The transition occurs only when the VOQs selected by $\mathbf{H}$ satisfy the conditions below:

1) For any $(i,j) \in \mathbf{S} \cap \overline{\mathbf{S}'}$: the VOQ is selected by $\mathbf{H}$ and decides to change its schedule from 1 to 0, which happens with probability $\overline{p}_{ij}$.
2) For any $(k,l) \in \overline{\mathbf{S}} \cap \mathbf{S}'$: the VOQ is selected by $\mathbf{H}$ and decides to change its schedule from 0 to 1, which happens with probability p_{kl}. The head-of-line packet in VOQ_{kl} has a size of L'_{kl} with

probability $P_{kl}(L'_{kl})$.

3) For any $(u, v) \in (\mathbf{S} \cap \overline{\mathbf{B}}) \cap (\mathbf{S}' \cap \overline{\mathbf{B}}) \cap \mathbf{H}$: the VOQ finished a packet transmission in previous time slot, and even though selected by **H** it decides to keep its schedule, which occurs with probability p_{uv}. The length of the new packet being transmitted is L'_{uv} with probability $P_{uv}(L'_{uv})$.
4) For any $(x, y) \in \mathbf{H} \cap \overline{\mathbf{S} \cup \mathbf{S}'} \cap \overline{\mathcal{N}(\mathbf{S})}$: neither the VOQ or any of its neighbors was in the schedule of the previous time slot, and though selected by **H** it decides to stay inactive, which occurs with probability $\overline{p}_{xy}$. Since **H** is a feasible schedule and $\overline{\mathbf{S}} \cap \mathbf{S}' \in \mathbf{H}$, $\mathbf{H} \cap \mathcal{N}(\overline{\mathbf{S}} \cap \mathbf{S}') = \emptyset$. Thus $\mathbf{H} \cap \overline{\mathbf{S} \cup \mathbf{S}'} \cap \overline{\mathcal{N}(\mathbf{S})}$ = $\mathbf{H} \cap \overline{\mathbf{S} \cup \mathbf{S}'} \cap \overline{\mathcal{N}(\mathbf{S} \cup \mathbf{S}')}$. We replace $\mathbf{H} \cap \overline{\mathbf{S} \cup \mathbf{S}'} \cap \overline{\mathcal{N}(\mathbf{S})}$ by $\mathbf{H} \cap \overline{\mathbf{S} \cup \mathbf{S}'} \cap \overline{\mathcal{N}(\mathbf{S} \cup \mathbf{S}')}$ in Eq. (6) for the proof of the stationary distribution in the following.

Since **H** is a feasible schedule, for any two VOQs in **H**, they are not neighbors of each other. Therefore, they can make the scheduling decisions independently. We then can multiply the probabilities of all the four categories above, which leads to the transition probability given by Eq. (6). **QED** ■

As we can see from Lemma 2, the state $\mathbf{Y}'$ only depends on the schedule of the previous time slot **Y**. Thus **Y**(n-1), **Y**(n), **Y**(n+1)$\cdots$ is a Markov chain, and the state transition probability is given in Eq. (6). We will derive the stationary distribution in the following.

Let us associate each VOQ of a switch with a non-negative weight $w_{ij}(n)$ (i.e. $w_{ij}(n) = loglog(Q_{ij}(n))$) at time n and define the probability $p_{ij} = \frac{e^{w_{ij}(n)}}{e^{w_{ij}(n)}+1}$. We have the following result.

Theorem 1: The Markov chain of the system has the following product-form stationary distribution:

$$\pi(\mathbf{Y}) = \frac{1}{\mathcal{Z}} \prod_{(i,j)\in\mathbf{S}} \frac{p_{ij}}{\overline{p}_{ij}} P_{ij}(L_{ij}) = \frac{1}{\mathcal{Z}} \prod_{(i,j)\in\mathbf{S}} e^{w_{ij}(n)} P_{ij}(L_{ij}), \tag{7}$$

where $\mathcal{Z}$ is the normalizing term such that $\sum_{\mathbf{Y}} \pi(\mathbf{Y}) = 1$:

$$\mathcal{Z} = \sum_{\mathbf{Y}\in\mathcal{Y}} \prod_{(i,j)\in\mathbf{S}} \frac{p_{ij}}{\overline{p}_{ij}} P_{ij}(L_{ij}) = \sum_{\mathbf{Y}\in\mathcal{Y}} \prod_{(i,j)\in\mathbf{S}} e^{w_{ij}(n)} P_{ij}(L_{ij}). \tag{8}$$

Proof: Define the probability:

$$\begin{aligned} Q(\mathbf{Y}', \mathbf{Y}) \quad &= \sum_{\mathbf{H}:\mathbf{S}\triangle\mathbf{S}'\in\mathbf{H}} a(\mathbf{H}) \prod_{(i,j)\in\mathbf{S}\cap\overline{\mathbf{S}}'} p_{ij} \times P_{ij}(L_{ij}) \prod_{(k,l)\in\overline{\mathbf{S}}\cap\mathbf{S}'} \overline{p}_{kl} \\ &\bullet \prod_{(u,v)\in(\mathbf{S}\cap\overline{\mathbf{B}})\cap(\mathbf{S}'\cap\overline{\mathbf{B}})\cap\mathbf{H}} p_{uv} \times P_{uv}(L_{uv}) \prod_{(x,y)\in\mathbf{H}\cap\overline{\mathbf{S}\cup\mathbf{S}'}\cap\overline{\mathcal{N}(\mathbf{S}\cup\mathbf{S}')}} \overline{p}_{xy}, \end{aligned} \tag{9}$$

Note that $Q(\mathbf{Y}', \mathbf{Y})$ is very similar to P(**Y**, **Y'**), and it can be considered as the transition probability of the time-reversed Markov chain. It is easy to verify that:

$$\pi(\mathbf{Y})P(\mathbf{Y},\mathbf{Y}') = \pi(\mathbf{Y}')Q(\mathbf{Y}',\mathbf{Y}). \tag{10}$$

Therefore,

$$\sum_{\mathbf{Y}} \pi(\mathbf{Y}) \cdot P(\mathbf{Y},\mathbf{Y}') = \sum_{\mathbf{Y}} \pi(\mathbf{Y}') \cdot Q(\mathbf{Y}',\mathbf{Y}) = \pi(\mathbf{Y}') \cdot \sum_{\mathbf{Y}} Q(\mathbf{Y}',\mathbf{Y}) = \pi(\mathbf{Y}') \tag{11}$$

The probability of state $\mathbf{Y}'$ is invariant over time, which means the distribution shown in Eq. 7 is "stationary". ■

D. System Stability

One of the most popular algorithm which has been proved stable is the *Maximum Weight Matching (MWM)* algorithm. The **MWM** algorithm selects a feasible schedule with the maximum weight:

$$\mathbf{S}^*(n) = \arg\max_{\mathbf{S}\in\mathcal{S}} \sum_{(i,j)\in\mathbf{S}} w_{ij}(n). \tag{12}$$

We can define the weight on a state $\mathbf{Y} = (\mathbf{S},\mathbf{L},\mathbf{R})$ as:

$$W(\mathbf{Y}) = W(\mathbf{S}) = \sum_{(i,j)\in\mathbf{S}} S_{ij}(n) w_{ij}(n). \tag{13}$$

The **MWM** algorithm has been widely studied and it has been proved to stabilize the system. For **MWM**, the result below has been established in [6].

Lemma 3: For a scheduling algorithm, if given any ϵ and δ such that $0 \le \epsilon$, $\delta < 1$, there exists a $B > 0$ such that the scheduling algorithm satisfies the condition: in any time slot t, with a probability greater than $1-\delta$, the scheduling algorithm can choose a schedule $\mathbf{S} \in \mathcal{S}$ which satisfies the following condition:

$$\sum_{(i,j)\in\mathbf{S}(n)} w_{ij}(n) \ge (1-\epsilon) \sum_{(k,l)\in\mathbf{S}^*(n)} w_{kl}(n), \tag{14}$$

whenever $||\mathbf{Q}(n)|| \ge B$, where $\mathbf{Q}(n) = (Q_{ij}(n))$ and $||\mathbf{Q}(n)|| = \left(\sum_{i,j} Q_{ij}^2(n)\right)^{1/2}$. Then the scheduling algorithm can stabilize the system.

Since we already derived the stationary distribution of the Markov chain, we will prove the system stability using Lemma 3. Before the proof of Theorem 3, we first have to prove the lemmas below.

Lemma 4: Suppose that $T(\cdot)$ is a function defined on a set $\mathcal{Y}$. For any probability distribution μ on $\mathcal{Y}$, define the function:

$$F(\mu, T(\mathbf{Y})) = E_\mu[T(\mathbf{Y})] + H(\mu), \tag{15}$$

where $H(\mu)$ is the entropy: $H(\mu) = -\sum_{\mathbf{Y}\in\mathcal{Y}} \mu(\mathbf{Y}) \log \mu(\mathbf{Y})$. Then $F(\cdot)$ is uniquely maximized by the distribution:

$$\mu^*(\mathbf{Y}) = \frac{1}{Z}\exp(T(\mathbf{Y})), \tag{16}$$

where $Z = \sum_{\mathbf{Y}\in\mathcal{Y}} \exp(T(\mathbf{Y}))$.

Proof: For any probability distribution μ, we have:

$$\begin{aligned}
F(\mu, T(\mathbf{Y})) &= E_\mu[T(\mathbf{Y})] + H(\mu) \\
&= \sum_{\mathbf{Y}\in\mathcal{Y}} \mu(\mathbf{Y})T(\mathbf{Y}) - \sum_{\mathbf{Y}\in\mathcal{Y}} \mu(\mathbf{Y}) \log \mu(\mathbf{Y}) \\
&= \sum_{\mathbf{Y}\in\mathcal{Y}} \mu(\mathbf{Y})(\log \mu^*(\mathbf{Y}) + \log Z) - \sum_{\mathbf{Y}\in\mathcal{Y}} \mu(\mathbf{Y}) \log \mu(\mathbf{Y}) \\
&= \sum_{\mathbf{Y}\in\mathcal{Y}} \mu(\mathbf{Y}) \log Z + \sum_{\mathbf{Y}\in\mathcal{Y}} \mu(\mathbf{Y}) \log \frac{\mu^*(\mathbf{Y})}{\mu(\mathbf{Y})} \\
&\leq \log Z \sum_{\mathbf{Y}\in\mathcal{Y}} \mu(\mathbf{Y}) + \log\Big(\sum_{\mathbf{Y}\in\mathcal{Y}} \mu(\mathbf{Y})\frac{\mu^*(\mathbf{Y})}{\mu(\mathbf{Y})}\Big) \text{ (from Jensen's inequality)} \\
&= \log Z,
\end{aligned} \tag{17}$$

with equality holding only when $\mu = \mu^*$. **QED** ■

Note that when $T(\mathbf{Y}) = 0$, uniform distribution maximizes $F(\mu, 0)$, and we have:

$$F(\mu, 0) = H(\mu) \leq \log Z = \log |\mathcal{Y}|. \tag{18}$$

Lemma 5: Let $W(\cdot)$ be the weight function and $W^*(\mathbf{S})$ the maximum weight. Define the set:

$$\mathcal{K} = \{\mathbf{Y} \in \mathcal{Y} : W(\mathbf{Y}) = W(\mathbf{S}) \leq (1-\epsilon)W^*(\mathbf{S})\}. \tag{19}$$

Then, we have:

$$\pi(\mathcal{K}) \leq \frac{\log|\mathcal{Y}| + |G_{min}|}{\epsilon W^*(\mathbf{S})}, \tag{20}$$

where $\pi(\mathcal{K})$ is the probability that a state $\mathbf{Y}$ is in the set $\mathcal{K}$ and $|G_{min}|$ is a constant we will define below.

Proof: As shown in Eq. (7), for a state $\mathbf{Y} = (\mathbf{S}, \mathbf{L}, \mathbf{R})$, its stationary distribution is:

$$\pi(\mathbf{Y}) = \frac{1}{\mathcal{Z}} \prod_{(i,j)\in\mathbf{S}} e^{(w_{ij}(n))} P_{ij}(L_{ij}) = \frac{1}{\mathcal{Z}} e^{W(\mathbf{S})} \prod_{(i,j)\in\mathbf{S}} P_{ij}(L_{ij}). \tag{21}$$

Define $G(\mathbf{Y})$ as:

$$G(\mathbf{Y}) = \ln \prod_{(i,j)\in\mathbf{S}} P_{ij}(L_{ij}) \tag{22}$$

Then we have $\pi(\mathbf{Y}) = \frac{1}{\mathcal{Z}} e^{W(\mathbf{Y})+G(\mathbf{Y})}$. Let G_{max} represent $\max_{\mathbf{Y}\in\mathcal{Y}} G(\mathbf{Y})$, and it is easy to see that $G_{max} \leq 0$ from Eq. (22). Let G_{min} represent $\min_{\mathbf{Y}\in\mathcal{Y}} G(\mathbf{Y})$. Then $|G_{min}| \geq |G(\mathbf{Y})|$.

According to Lemma 4, π maximizes $F\Big(\mu, T(\mathbf{Y}) = W(\mathbf{Y}) + G(\mathbf{Y})\Big)$.

Let $\mathbf{Y}^* = (\mathbf{S}^*, \mathbf{L}^*, \mathbf{R}^*)$ be a state which has the maximum weight. Let π' be the distribution that assigns all probability on $\mathbf{Y}^*$ such that:

$$\pi'(\mathbf{Y}) = \begin{cases} 1 & \text{if } \mathbf{Y} = \mathbf{Y}^* \\ 0 & \text{otherwise} \end{cases}$$

Then we have:

$$\begin{aligned} F(\pi', W(\mathbf{Y}) + G(\mathbf{Y})) &= E_{\pi'}[W(\mathbf{Y}) + G(\mathbf{Y})] + H(\pi') \\ &= W^*(\mathbf{S}) + G(\mathbf{Y}^*) + H(\pi') \\ &\leq F(\pi, W(\mathbf{S}) + G(\mathbf{Y})) = E_{\pi}[W(\mathbf{Y}) + G(\mathbf{Y})] + H(\pi) \\ &= \sum_{\mathbf{Y}\in\mathcal{K}} \pi(\mathbf{Y})\mathbf{W}(\mathbf{Y}) + \sum_{\mathbf{Y}\in\overline{\mathcal{K}}} \pi(\mathbf{Y})\mathbf{W}(\mathbf{Y}) + E_{\pi}[G(\mathbf{Y})] + H(\pi) \\ &\leq \pi(\mathcal{K})(1-\epsilon)W^*(\mathbf{S}) + (1-\pi(\mathcal{K}))W^*(\mathbf{S}) + G_{max} + H(\pi) \\ &\leq W^*(\mathbf{S})(1-\epsilon\pi(\mathcal{K})) + G_{max} + H(\pi) \end{aligned} \tag{23}$$

So,

$$\begin{aligned} W^*(\mathbf{S}) + G(\mathbf{Y}^*) + H(\pi') &\leq W^*(\mathbf{S})(1-\epsilon\pi(\mathcal{K})) + G_{max} + H(\pi) \\ \epsilon\pi(\mathcal{K})W^*(\mathbf{S}) &\leq H(\pi) - H(\pi') + G_{max} - G(\mathbf{Y}^*) \\ &\leq H(\pi) + |G_{min}| \leq \log|\mathcal{Y}| + |G_{min}| \\ \pi(\mathcal{K}) &\leq \frac{\log|\mathcal{Y}| + |G_{min}|}{\epsilon W^*(\mathbf{S})} \end{aligned} \tag{24}$$

■

Theorem 2: The scheduling algorithm can stabilize the system if the input traffic is admissible.

Proof: For any $\delta > 0$, we have $\pi(\mathcal{K}) < \delta$, if the maximum weight satisfies the condition:

$$W^*(\mathbf{S}) > \frac{\log|\mathcal{Y}| + |G_{min}|}{\epsilon\delta} > \frac{\log|\mathcal{Y}| + |G_{min}|}{\epsilon\delta}. \tag{25}$$

So, for any $\epsilon, \delta > 0$, there exists a $B > 0$ such that whenever $||\mathbf{Q}(n)|| > B$, Eq. (25) holds and then $\pi(\mathcal{K}) < \delta$. Hence the scheduling algorithm can stabilize the system according to Lemma 3. ■

E. Computation Complexity

At the beginning of each time slot, we have to generate a feasible schedule $\mathbf{H}(n)$ using Hamiltonian walk. There are N crosspoints (i,j) such that $H_{ij}(n) = 1$. According the algorithm, the scheduler has to check all these N VOQs and decide whether to change their scheduling or not. Thus, the computation complexity is $O(N)$. By making the following simple modification, we can reduce the computation complexity from $O(N)$ to $O(1)$.

As mentioned above, the reason that A-MWM has a complexity of $O(N)$ is because there are N VOQs in $\mathbf{H}(n)$. If we can generate a schedule $\mathbf{H}(n)$, where there is only one (i,j) such that $H_{ij}(n) = 1$, the scheduler then only has to decide whether to change the scheduling of one VOQ at each time slot. Thus, the computation complexity is reduced to $O(1)$. For a $N \times N$ switch, there are N^2 VOQs totally. Therefore, there are N^2 schedules such that only one VOQ is in each schedule. The Hamiltonian walk $\mathbf{H}(n)$ can visit all these N^2 schedules once during every N^2 slots.

The modified version of A-MWM still can achieve 100% throughput for any admissible i.i.d arrival traffic. The only difference is that it takes a longer time for the system to converge. For example, if there are N VOQs selected by $\mathbf{H}(n)$, all these N VOQs may change their schedules in one slot. But if there is only one VOQ in $\mathbf{H}(n)$, it takes about N slots to have the same schedule changes. Therefore, the $O(1)$ version of A-MWM takes a longer time to converge than the $O(N)$ version.

Now, we give a brief proof of the system stability for the modified version of A-MWM. Following the new algorithm, at each time slot, there is only one VOQ selected by the Hamiltonian walk schedule **H**(n). Therefore, Lemma 1 should be changed to:

Lemma 6: A state $\mathbf{Y} = (\mathbf{S}, \mathbf{L}, \mathbf{R})$ can transit to a state $\mathbf{Y}' = (\mathbf{S}', \mathbf{L}', \mathbf{R}')$ if and only if $\mathbf{Y}'$ satisfies the conditions below:

1) If $(i,j) \in \mathbf{B}$, $S'_{ij} = S_{ij} = 1$, $L'_{ij} = L_{ij}$, and $R'_{ij} = R_{ij} - 1$.

2) $\mathbf{S} \cup \mathbf{S}' \in \mathcal{S}$.

3) There exists at most only one (i, j) such that $(i, j) \in \mathbf{S}$ and $(i, j) \notin \mathbf{S}'$, or $(i, j) \notin \mathbf{S}$ and $(i, j) \in \mathbf{S}'$.

Proof of Lemma 6 is almost the same as Lemma 1. Condition 3) is necessary here since there is only one VOQ in $\mathbf{H}$(n). Therefore, in each transition, at most only one VOQ can change its schedule. When two states $\mathbf{Y}$ and $\mathbf{Y}'$ satisfy the conditions above, the system can transit from $\mathbf{Y}$ to $\mathbf{Y}'$. The transition probability is the same as Eq. (6). Note that now there is at most one (i, j) in $\mathbf{S} \triangle \mathbf{S}'$ = $(\mathbf{S} \cap \overline{\mathbf{S}'}) \cup (\overline{\mathbf{S}} \cap \mathbf{S}')$, and $a(\mathbf{H}) = \frac{1}{N^2}$.

Since the transition probability does not change, following the proof in II-C, we can also derive that the stationary distribution of the system under the modified A-MWM still has a product-form solution. The system stability then can be proved following the same methodology.

III. Simulations

We have run simulations for different scenarios to measure the delay performance of our algorithm. Different traffic arrival patterns are studied, including uniform and non-uniform. We also study scenarios with different packet size distributions. We compare the delay performance of A-MWM with two algorithms proposed in [2]: *Packet MWM* (P-MWM) and *Packet Incremental MWM* (PI-MWM).

A. P-MWM and PI-MWM

The packet MWM scheduling algorithm generates a new schedule using MWM algorithm whenever either of the following conditions hold:

- All packet transmissions end at the same time.
- All the queues selected for transfer become empty.

The packet incremental MWM works as follows:

- Whenever either all packet transmissions end at the same time, or all the queues selected for transfer become empty, a new schedule is generated using the MWM algorithm.
- Whenever some queues selected for transfer become idle (i.e., either they are empty, or packet transmissions end) a partial update of the schedule is allowed, according to an MWM algorithm among idle ports.

Using P-MWM algorithm, after a schedule is generated, the system may have to use that schedule for many time slots until either of the two conditions mentioned above holds. Therefore, the schedule reconfiguration of P-MWM is not frequent, but each reconfiguration still requires a computation complexity of $O(N^3)$. As shown in the results below, this negatively affects the performance of P-MWM, due to the fact that several time slots elapse before a new schedule is generated (i.e., some queues selected for transfer become idle).

PI-MWM can have a much better performance, since it has to update the schedule at every time slot among idle ports. So, the computation complexity of PI-MWM is the same as MWM, which is $O(N^3)$. It is not practical for real systems, but it can guarantee good delay performance and thus serves as a benchmark.

B. Packet Size Distribution

In the simulation, we consider three different packet size distributions.

1) Uniform(a, b). Packet sizes are uniformly distributed between a and b. As in [2], we use $a = 1$ and $b = 192$ in the following.
2) Bimodal(a, b; p_a). Packet sizes are chosen as a cells with probability p_a, or b cells with probability $1 - p_a$. In the following, we set $a = 3$, $b = 100$ and $p_a = 0.5$.
3) Exponential(a, b). Packet sizes are exponentially distributed over $[a, b]$, following the truncated Pareto distribution:
$$P(l) = \frac{c}{l^\beta},\ l = \text{a, a+1, ... , b}, \tag{26}$$
where l is the packet length, β is the Pareto parameter and c is the normalization constant. In the simulation $\beta = 1.7$, $a = 1$ and $b = 100$.

C. Uniform Scenario

For uniform traffic, a new cell is destined uniformly for all output ports. Let λ represent the traffic load, the arrival rate between input i and output j is $\lambda_{ij} = \frac{\lambda}{N}$. The delay performance comparisons are shown in Fig. 2 and Fig. 3. In Fig. 2, packet sizes are uniformly distributed over $[1, 192]$. We can see that even though our algorithm has a much lower computation complexity, the delay is quite close to that of PI-MWM, especially when the load is light. For P-MWM, the delay performance is poor for the reason that we discussed in the previous subsection. As the load increases, the delay of P-MWM increases very fast. In Fig. 3, packet sizes are exponentially distributed over $[1, 100]$. The average packet size now is around 9. PI-MWM can provide very good delay performance compared to our algorithm. But as shown

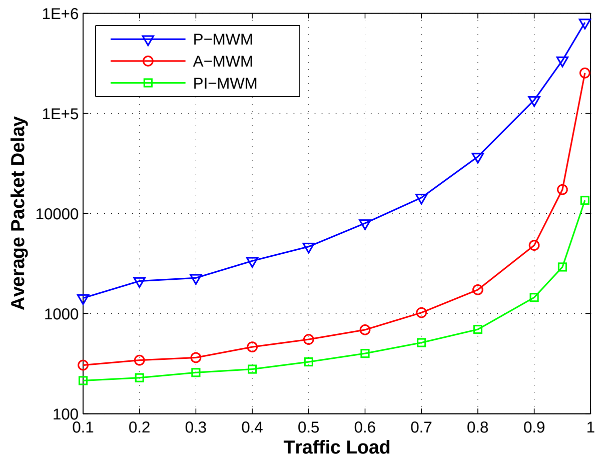

Fig. 2: N=16, uniform traffic, uniform packet size distribution

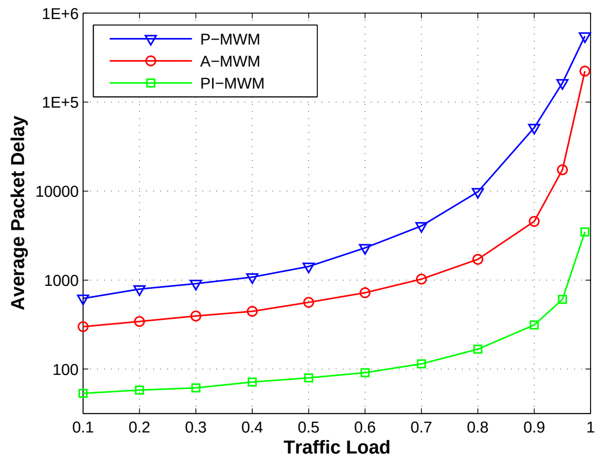

Fig. 3: N=16, uniform traffic, exponential packet size distribution

in the figure, the delay of our algorithm is still much less than that of P-MWM, especially when the load is heavy ($0.6 < \lambda < 0.95$).

D. *Non-Uniform Scenario*

We run the simulations for the following traffic patterns:

- Lin-diagonal: Arrival rates at the same input differ linearly, i.e, $\lambda_{i(i+j \pmod N)} - \lambda_{i(i+j+1 \pmod N)} = 2\lambda/N(N+1)$.
- Hot-spot: For input port i, $\lambda_{ii} = \omega\lambda$ and $\lambda_{ij} = (1-\omega)\lambda/(N-1)$, for $i \neq j$. We can get different traffic patterns by varying the hot-spot factor ω.

The delay performance for hot-spot and lin-diagonal traffic are shown in 4 and Fig. 5, respectively. For bimodal distribution, the delay of our algorithm is very close to the PI-MWM. But for exponential packet

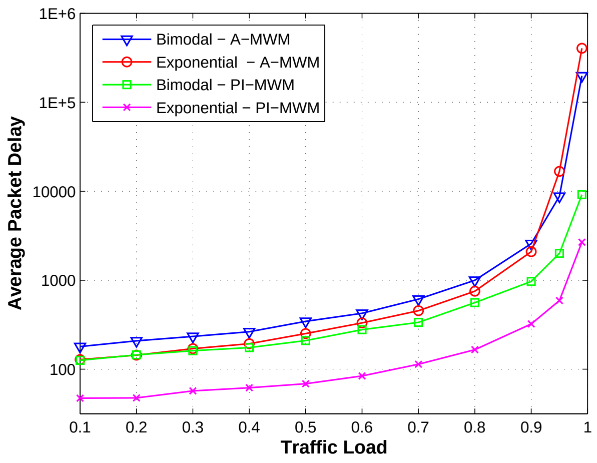

Fig. 4: Switch size N=16, hot-spot traffic

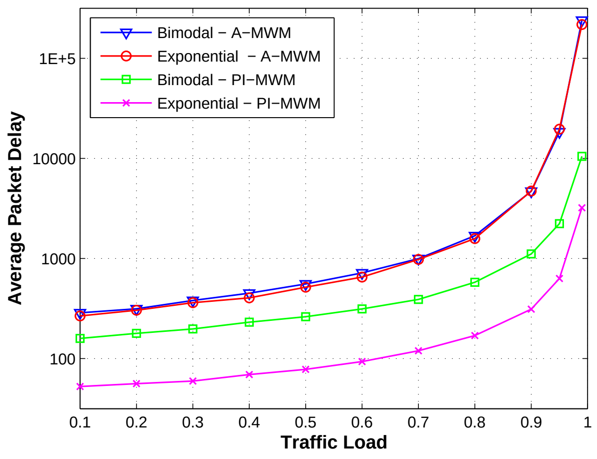

Fig. 5: Switch size N=16, lin-diagonal traffic

size distribution, the delay of our algorithm is much larger.

One interesting result is that for different packet size distributions, our algorithm can provide quite similar delay performance. As you can see in Fig. 5, the delay for bimodal and exponential distributions are almost the same.

IV. Buffered Crossbar Switch

In this section, we show how the algorithm we proposed can be implemented in a distributed manner for a buffered crossbar switch.

A. *System Model*

In a buffered crossbar switch, each crosspoint has a buffer as shown in Fig. 6. We assume that each crosspoint buffer has a size of one cell. Let CB_{ij} denote the buffer at the crosspoint between input i and

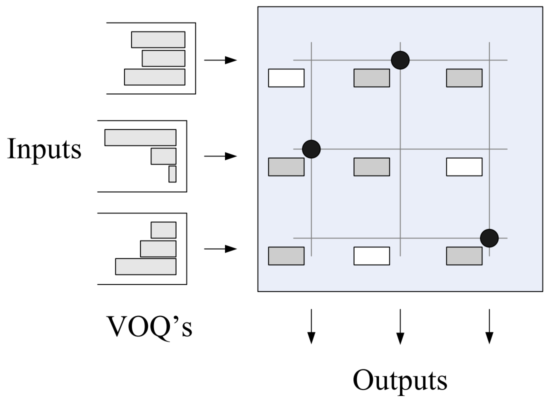

Fig. 6: Buffered Crossbar Switch

output j. $B_{ij}(n) \in \{0,1\}$ denotes the occupancy of CB_{ij} at time n.

Due to the existence of the crosspoint buffers, the input and output schedulers can be independent. Now, a schedule can be represented by **S(n)** = [$\mathbf{S}^I(n), \mathbf{S}^O(n)$]. $\mathbf{S}^I(n) = [S^I_{ij}(n)]$ is the input schedule. Each input port can only transmit at most one cell at each time slot. Thus the input schedule is subject to the following constraints:

$$\sum_j S^I_{ij}(n) \le 1,\ S^I_{ij}(n) = 0 \text{ if } B_{ij}(n) = 1. \tag{27}$$

$\mathbf{S}^O(n) = [S^O_{ij}(n)]$ is the output schedule. It has to satisfy the following constraints:

$$\sum_i S^O_{ij}(n) \le 1,\ S^O_{ij}(n) = 0 \text{ if } B_{ij}(n) = 0. \tag{28}$$

B. P-DISQUO Schedule

As in our previous work [7], we define a P-DISQUO schedule:

Definition 5: A ***P-DISQUO schedule*** $\mathbf{X}(n)$ is an $N \times N$ matrix, where $X_{ij}(n) \in \{0,1\}$, and $\sum_i X_{ij}(n) \le 1$, $\sum_j X_{ij}(n) \le 1$.

A P-DISQUO schedule **X** then has the following property:

Property 1: If $(i,j) \in$ **X**, $\forall (k,l) \in \mathcal{N}(i,j)$, $(k,l) \notin$ **X**.

We also define the P-DISQUO schedule to have the following properties:

Property 2: At each time slot, when a P-DISQUO schedule is generated, each input and output port determine their schedules by following the rules below:

- For input i, when $X_{ij}(n) = 1$, if $Q_{ij}(n) > 0$ and $B_{ij}(n-1) = 0$, then $S^I_{ij}(n) = 1$. Otherwise, $S^I_{ij}(n) = 0$.

- For output j, if $X_{ij}(n) = 1$ and $B_{ij}(n) > 0$, $S^{O}_{ij}(n) = 1$.

Property 3: For an input i, if $\forall j$, $X_{ij} = 0$, then it is referred to as a *free input*. A free input port can randomly pick an eligible crosspoint to serve, i.e. it can transfer a packet to any free crosspoint buffer.

Property 4: For an output port j, if $\forall i$, $X_{ij} = 0$, then it is a *free output*. A free output can randomly pick a non-empty crosspoint to serve.

Let $\mathcal{X}$ represent the set of all P-DISQUO schedules.

C. P-DISQUO Scheduling Algorithm

In the algorithm, each input i only needs to keep track of the P-DISQUO schedule in the previous slot, i.e. for which output j was $X_{ij}(n-1) = 1$. Similarly, each output only needs to keep track of for which input i was $X_{ij}(n-1) = 1$. Since the algorithm is distributed, there is no message passing between inputs and outputs. The algorithm has to make sure that if $X_{ij}(n) = 1$, both input i and output j are aware of this. Then the inputs and outputs can keep a consistent view of the P-DISQUO schedule.

Input Scheduling Decisions

```
At each input port $i$, assume $(i, j)$ is selected by $\mathbf{H(n)}$.
∘ If there exists a $j'$, with $X_{ij'}(n-1) = 1$:
    - If $j = j'$, $(i, j) \in \mathbf{X}(n-1)$ and $(i, j) \in \mathbf{H}(n)$, and the transmission between input $i$ and
output $j$
      is completed:
        (a) $X_{ij}(n) = 1$ with probability $p_{ij}$;
        (b) $X_{ij}(n) = 0$ with probability $\overline{p}_{ij} = 1 - p_{ij}$.
    - Else,
        (c) $X_{ij}(n) = 0$.
∘ Else, if there is no $j'$ such that $X_{ij'}(n-1) = 1$, then input $i$ is a free input:
    - If $\forall (k, l) \in \mathcal{N}(i, j)$, $X_{kl}(n-1) = 0$ (We will explain later how an input can learn this):
        (d) $X_{ij}(n) = 1$ with probability $p_{ij}$;
        (e) $X_{ij}(n) = 0$ with probability $\overline{p}_{ij} = 1 - p_{ij}$.
    - Else,
        (f) $X_{ij}(n) = 0$.
```

Output Scheduling Decisions

Each output port j has to learn the scheduling decision made by the input. Assume (i, j) is selected by $\mathbf{H}(n)$.

∘ If there exists an i', with $X_{i'j}(n-1) = 1$:

- If $i = i'$, $(i, j) \in \mathbf{X}(n-1)$ and $(i, j) \in \mathbf{H}(n)$. As shown above, input i may change X_{ij} from 1 to 0. Therefore, output j has to observe the crosspoint buffer to learn the input's decision.

  (a) If input i transmits a packet to CB_{ij} at the beginning of time n, $X_{ij}(n) = 1$

  (b) Otherwise, $X_{ij}(n) = 0$.

- Else,

  (c) $X_{ij}(n) = X_{ij}(n-1) = 0$

∘ Else, if there is no i' such that $X_{i'j}(n-1) = 1$, then output j is free:

- If the buffer at crosspoint (i, j) is empty and input i sends a packet to CB_{ij} at the beginning of time n, or if the buffer is not empty, output j has to transmit this packet from CB_{ij} at time n and if then input i sends a packet to CB_{ij} at the beginning of time $n+1$, output j can update its schedule of time n as:

  (d) $X_{ij}(n) = 1$.

- Else,

  (e) $X_{ij}(n) = 0$.

As we can see that in the algorithm above, the inputs are making the scheduling decisions and updating the P-DISQUO schedule based on $\mathbf{H}(n)$. The output ports have to learn the inputs' decisions. The key point of P-DISQUO is that by observing crosspoint buffers, an input and an output can learn each other's decisions implicitly. For example, if VOQ_{ij} is selected by $\mathbf{H}(n)$, and free input i decides to set $X_{ij}(n) = 1$ from $X_{ij}(n-1) = 0$, it has to make sure that output j was also free. Input i can learn whether output j was free or not by observing the crosspoint buffer CB_{ij} at time n. If it is served by output port j at time n, input i learns that the output was free at time $n-1$.

Theorem 3: The P-DISQUO scheduling algorithm can stabilize the system if the input traffic is admissible.

Proof: The P-DISQUO schedule essentially is a feasible schedule in an input-queued switch. So the system stability can also be proved following the proof in an input-queued switch. ■

V. Simulations

In this section, we compare the delay performance of our distributed scheduling algorithm (DS) with output-queued (OQ) switches, which require a speedup of N but have the best performance and thus serves as a lower bound. We run extensive simulations for different traffic patterns with different packet size distributions. The results are presented below.

A. Uniform Scenario

The delay performance comparison is shown in Fig. 7. We can see that even though we do not require any speedup, the packet delay of our algorithm is very close to the output-queued switch (OQ), for different packet size distributions.

B. Non-Uniform Scenario

The delay performance for hot-spot traffic is shown in 8. We can see that the delay performance of our algorithm is still very close to the output-queued switch for non-uniform arrivals. When the traffic load is less than 0.7, the delay performance of our algorithm is just a little higher than the output-queued switches. When the traffic is heavy ($\lambda > 0.7$), the delay increases but still close to the bound.

Note that for non-uniform traffic, simple algorithms such as dual round robin [8] can not stabilize the system even for cell-mode scheduling. Our algorithm not only can stabilize the system for non-uniform variable-sized packets, but also can provide good delay performance. It can provide consistent service for different traffic patterns, which makes it very practical to implement in real systems.

C. Impact of Switch Size

In this section, we will study the impact of switch size on the delay performance. Generally, for input-queued switch, the average delay increases linearly with the switch size. For output-queued switches, delay is independent of the size. Fig. 9 shows the delay performance of switches with different sizes under hot-spot traffic (ω is 0.5) with uniform packet distribution. We can see that, the delay is almost the same for different switch sizes. As the size increases, the delay even decreases slightly when the traffic becomes heavy ($\lambda > 0.8$). This shows that our algorithm is scalable and could be implemented in switching systems with a large number of ingress and egress ports.

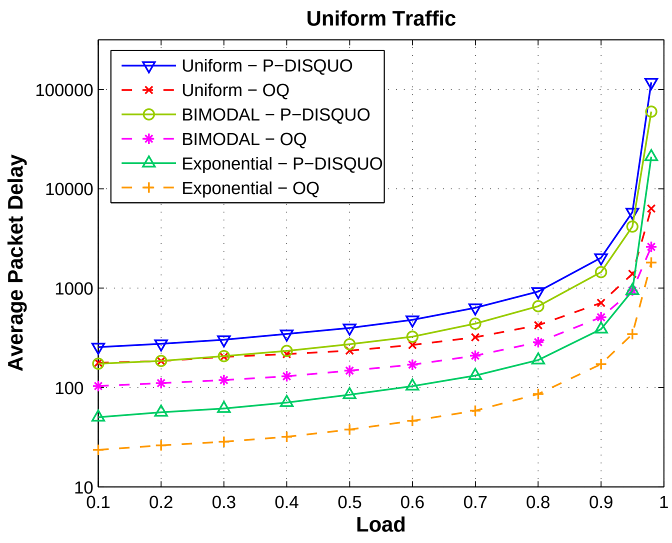

Fig. 7: Switch size N=16, uniform traffic

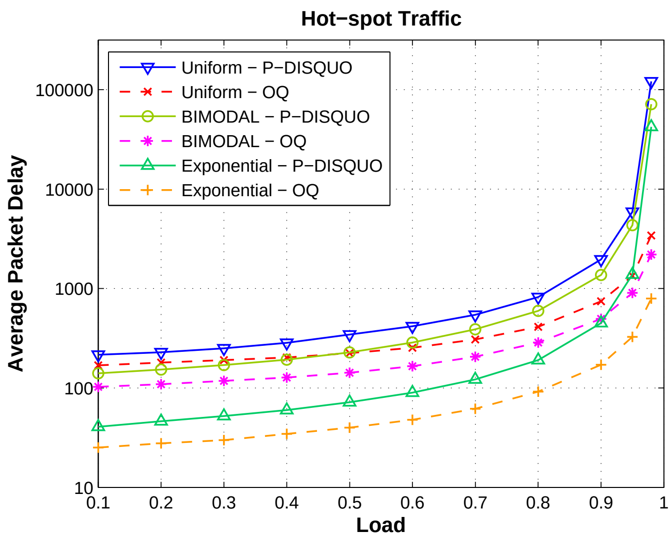

Fig. 8: Switch size N=16, hot-spot traffic, w=0.5

VI. Conclusion

In this paper, we propose a low complexity packet-mode scheduling algorithm (A-MWM) for an input-queued switch. The complexity of the algorithm is $O(N)$, and can be reduced to $O(1)$ easily. We prove that it can achieve 100% throughput under any admissible ON-OFF arrival traffic with any finite packet size distribution. We then extend the idea to a distributed algorithm (P-DISQUO) for a buffered crossbar switch. When all packets have unit size, our algorithm becomes a low-complexity cell-mode scheduling algorithm. Our simulation results show that it can provide very good delay performance under different traffic arrivals. The results also show that in a buffered crossbar switch, packet delay is not dependent on

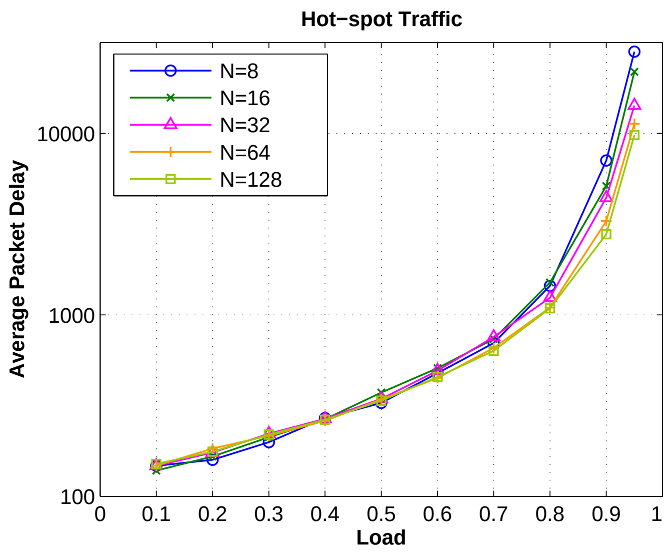

Fig. 9: Impact of switch size, hot-spot traffic, w=0.5

the switch size, which means that the distributed algorithm can scale with the number of switch ports. The algorithm has low implementation cost, provides good delay performanc, and is very scalable.

References

[1] L. Tassiullas and A. Ephremides, "Stability Properties of Constrained Queuing Systems and Scheduling Policies for Maximum Throughput in Multihop Radio Networks," *IEEE Transactions on Automatic Control*, vol. 37, pp. 1936–1949, December 1992.

[2] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Packet-Mode Scheduling in Input-Queued Cell-Based Switches," *IEEE Transactions on Networking*, vol. 10, October 2002.

[3] Y. Ganjali, A. Keshavarzian, and D. Shah, "Input Queued Switches: Cell Switching vs. Packet Switching," in *Proc. of IEEE INFOCOM*, (San Francisco, California), 2003.

[4] P. Giaccone, B. Prabhakar, and D. Shah, "Toward Simple, High Performance Schedulers for High-Aggregate Bandwidth Switches," in *Proc. of IEEE INFOCOM*, (New York), 2002.

[5] A. Nijenhuis and H. S. Wilf, *Combinatorial Algorithms for Computers and Calculators* . Academic Press, 1978.

[6] A. Eryilmaz, R. Srikant, and J. R. Perkins, "Stable Scheduling Policies for Fading Wireless Channels," *IEEE/ACM Transactions on Networking*, vol. 13, pp. 411–424, April 2005.

[7] S. Ye, Y. Shen, and S. S. Panwar, "DISQUO: A Distributed 100% Throughput Algorithm for a Buffered Crossbar Switch," in *Proceedings of IEEE Workshop on High Performance Switching and Routing,*, June 2010.

[8] R. Rojas-Cessa, E. Oki, and H. J. Chao, "On the Combined Input-Crosspoint Buffered Packet Switch with Round-Robin Arbitration," *IEEE Transactions on Communications*, vol. 53, pp. 1945–1951, November 2005.