An O(1) Scheduling Algorithm for Variable-size Packet Switching Systems

Shunyuan Ye†, Yanming Shen‡, Shivendra Panwar†
ECE Department, Polytechnic Institute of NYU†
School of CST, Dalian University of Technology, China‡
e-mail: {sye02@students, syanming@photon, panwar@catt}.poly.edu

Abstract—**Internet traffic has increased at a very fast pace in recent years. The traffic demand requires that future packet switching systems should be able to switch packets in a very short time, i.e., just a few nanoseconds. Algorithms with lower computation complexity are more desirable for this high-speed switching design. Among the existing algorithms that can achieve 100% throughut for input-queued switches for any admissible Bernoulli traffic, ALGO3 [1] and EMHW [2] have the lowest computation complexity, which is $O(\log N)$, where N is the number of ports in the switch.**

In this paper, we propose a randomized scheduling algorithm, which can also stabilize the system for any admissible traffic that satisfies the strong law of large number. The algorithm has a complexity of $O(1)$. Since the complexity does not increase with the size of a switch, the algorithm is highly scalable and a good choice for future high-speed switch designs. We also show that the algorithm can be implemented in a distributed way by using a low-rate control channel. Simulation results show that the algorithm can provide a good delay performance as compared to algorithms with higher computation complexity.

I. Introduction

The seminal work by Tassiulas and Ephremides [3] initiated the study of scheduling algorithms in packet switching systems and wireless networks. *Maximum weight matching* (**MWM**), which was proposed in [3], can stabilize the system whenever the arrival traffic is admissible Bernoulli i.i.d.. However, the MWM algorithm is not practical due to its $O(N^3)$ complexity. A number of practical scheduling algorithms, such as iSLIP [4], iLQF [5] and DRRM [6], were proposed for implementation. However, when the arriving traffic is non-uniform, these algorithms cannot achieve 100% throughput, and induce a higher delay as compared to the MWM algorithm.

A randomized scheduling algorithm with $O(N)$ was proposed by Tassiulas [7]. This algorithm can reduce the complexity by utilizing the schedule of the previous time slot. It can achieve 100% throughput for any admissible Bernoulli traffic. But the delay performance is unacceptably high. Several algorithms were proposed by Giaccone et al. [1] to improve the delay performance. SERENA, which also considers new arrivals, besides using memory, has a much better delay performance than the one in [7]. But it also has a complexity of $O(N)$. Li et al. [2] proposed a scheduling algorithm using exhaustive service matching. The algorithm has a complexity of $O(\log N)$, thus it still grows with the size of a switch.

All these scheduling algorithms assume that variable-size packets are segmented into fixed-size cells after arriving. They are reassembled back into original packets at the output ports. Cells from a packet are switched independently, and 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.* [8], 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. Therefore, the reassembly of packets at the output ports is much easier and requires less memory and complexity, as shown in Fig. 1. They also showed that for some packet size distributions, packet-mode scheduling can have a better delay performance than the cell-mode scheduling. However, the algorithm proposed, called PI-MWM, is a modification of MWM. Therefore, it still has a high computation complexity. Ganjali *et al.* [9] show that PI-MWM is stable for any form of re-generative admissible traffic, rather than only Bernoulli i.i.d. traffic.

The objective of this paper is to design a scheduling algorithm with a low computation complexity and simple hardware implementation. The algorithm we proposed has a computation complexity of only $O(1)$. It can schedule variable-size packets in both packet mode and cell mode. We prove that the algorithm can stabilize the system for any admissible traffic that satisfies the strong law of large number (SLLN), which we believe is the first algorithm to achieve this with a complexity of $O(1)$. For practical hardware implementation, we also show that the algorithm can be implemented in a distributed way, by using a low-rate control channel to pass a one-bit message. This is in contrast to the previous implementations cited, all of which require a centralized scheduler. Simulation results are presented to compare our algorithm with previously proposed algorithms.

This paper is organized as follows. In Sec. II, we present the scheduling algorithm and prove the system stability. We then present simulation results in Sec. III and conclude the paper in Sec. IV.

II. Randomized Scheduling Algorithms

A. System Model

An $N \times N$ input-queued switch is shown in Fig. 1. 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

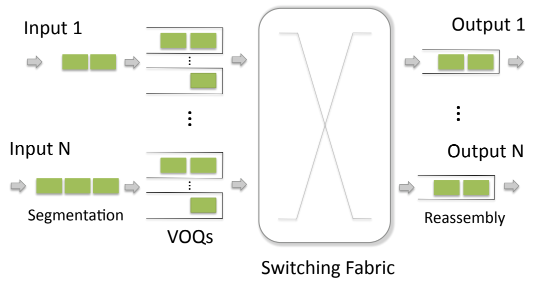

Fig. 1. Packet-Mode Input-Queued Switch

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.

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 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 the OFF state. The number of slots spent in OFF state is geometrically distributed. Let $A_{ij}(n)$ denote the number of cells that have arrived at input i for output j up to time n. We assume that $A_{ij}(0) = 0$, and the arrival process satisfies the *strong law of large number* (SLLN):

$$\lim_{n\to\infty} \frac{A_{ij}(n)}{n} = \lambda_{ij} \tag{1}$$

almost surely for any i and j. λ_{ij} is the arrival rate at VOQ_{ij}.

Definition 1. *An arrival process is said to be* strictly admissible *if it satisfies:*

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

Let $D_{ij}(n)$ denote the number of cells depart from VOQ_{ij} up to time n.

Definition 2. *A queueing system is said to be* rate stable *if with probability one,*

$$\lim_{n\to\infty} \frac{D_{ij}(n)}{n} = \lambda_{ij}, \tag{3}$$

for any i and j.

This is equivalent to:

$$\lim_{n\to\infty} \frac{Q_{ij}(n)}{n} = 0, \tag{4}$$

with probability one.

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 $S_{ij} = 1$. When $(i, j) \in \mathbf{S}$, we say crosspoint (i, j) is *active*, and input i is connected to output j so that one cell from a packet can be delivered from input i to output j. Otherwise, (i, j) is referred to as *inactive*.

Definition 3. *A* feasible schedule $\mathbf{S}(n)$ *is an $N \times N$ matrix, where $S_{ij}(n) \in \{0, 1\}$, and $\sum_i S_{ij}(n) \le 1$, $\sum_j S_{ij}(n) \le 1$.*

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 4. *For a crosspoint (i, j), its* neighbors *are defined as:*

$$\mathcal{N}(i, j) = \{(i', j) \text{ or } (i, j') \mid \forall i' \neq i, \forall j' \neq j\} \tag{5}$$

Therefore, 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, and $|\mathcal{S}|$ represent the size of the set $\mathcal{S}$. For a $N \times N$ input-queued switch, we have:

$$N! < |\mathcal{S}| < 2^{N\times N}, N \ge 2. \tag{6}$$

B. Randomized Scheduling Algorithm

Tassiulas first proposed a randomized scheduling algorithm with a complexity of $O(N)$ in [7] . The algorithm works as follows. Let $\mathbf{S}(n-1)$ represent the schedule at time $n-1$. At the beginning of time slot n, it randomly generates a schedule $\mathbf{R}(n)$, and then compare the weights of $\mathbf{S}(n-1)$ and $\mathbf{R}(n)$, where the weight of a schedule is defined as: $W(\mathbf{S}) = \sum_{i,j} S_{ij} \times Q_{ij}$. Then the new schedule is decided following the rule:

$$\mathbf{S}(n) = \arg \max_{\mathbf{S}\in\{\mathbf{S}(n-1),\mathbf{R}(n)\}} W(\mathbf{S})$$

.

As we can see, the algorithm is utilizing the memory of the schedule in the previous time slot $\mathbf{S}(n-1)$. Therefore, it can reduce the computational complexity to $O(N)$, and still achieve 100% throughput for any admissible Bernoulli traffic. However, its delay performance is very poor. Giaccone et al. [1] pointed out that instead of picking a schedule between $\mathbf{S}(n-1)$ and $\mathbf{R}(n)$, a new schedule can be generated by picking heavy edges from $\mathbf{S}(n-1)$ and $\mathbf{R}(n)$. This is referred as a *merge*. The new algorithm, which is called LAURA, can have a much better delay performance.

From this prior work, we can see that using memory can significantly reduce the computation complexity. The reason is that there is at most one arrival (departure) in a time slot for each input (output). A schedule with a heavy weight will continue to be heavy over a few time slots with a very high probability. Therefore, the previous schedule is likely to have high weight, which we can utilize. By using memory, the scheduling algorithm can be highly simplified and still maintain a good throughput performance.

Motivated by this observation, our algorithm is also designed to use memory. But unlike [7] and [1], our algorithm adds (or removes) an edge to (or from) the schedule with a probability, which is a function of the queue length. The system will converge to a steady state when schedules with heavier weights have higher probabilities to be selected at every time slot [10]–[14]. The randomized scheduling algorithm works as follows.

At the beginning of time n, generate a feasible schedule $\mathbf{H}(n)$, which satisfies the condition: $\sum_{i,j} H_{ij}(n) = 1$. This means that there is one, and only one, active crosspoint in $\mathbf{H}(n)$. For a $N \times N$ switch, there are totally N^2 such schedules. After $\mathbf{H}(n)$ is generated, the schedule $\mathbf{S}(n)$ is determined following the algorithm below.

```
Randomized Scheduling Algorithm
∘ ∀ $(i,j) \notin \mathbf{H}(n)$:
    (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 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) $S_{ij}(n) = 0$.
```

Let us associate each VOQ with a weight $w_{ij}(n) = W(Q_{ij}(n))$, which is a concave function of the queue size (i.e., $w_{ij}(n) = \log(\log(Q_{ij}(n)))$). The probability p_{ij} is defined as: $p_{ij} = \frac{e^{w_{ij}(n)}}{e^{w_{ij}(n)}+1}$. p_{ij} is a concave function of the queue length Q_{ij} such that a longer queue has a higher probability of receiving service.

C. Computational Complexity

Note that in our algorithm, $S_{ij}(n)$ can change its value only when (i,j) is in $\mathbf{H}(n)$. Since there is only one crosspoint (i,j) in $\mathbf{H}(n)$, the scheduler only has to decide whether to change the value of $S_{ij}(n)$ based on the transmission status of $\mathcal{N}(i,j)$ and the probability p_{ij}. Therefore the computation complexity of the algorithm is $O(1)$. The algorithm is designed to schedule variable-size packets. But when all the arriving packets have a fixed size of one cell, the algorithm also works as a cell-mode scheduling algorithm.

D. Stationary Distribution

Define the state:

$$\mathbf{Y} = (\mathbf{S}, \mathbf{L}, \mathbf{R}), \tag{7}$$

where $\mathbf{S}$ is the schedule. $\mathbf{L} = \{L_{ij}\}$, L_{ij} is the size of the packet being transmitted between input i and output j. $L_{ij} = 0, 1, 2 \cdots$, where $L_{ij} = 0$ indicates no packet is currently being transmitted. Let $P_{ij}(L_{ij})$ represent the probability that the head-of-line packet in VOQ_{ij} is of size L_{ij}. We assume that the distribution of packet lengths for each VOQ is fixed, namely, $P_{ij}(L_{ij})$ is a constant. Packet sizes are assumed to be multiples of a slot size. $\mathbf{R} = \{R_{ij}\}$, and R_{ij} is the remaining time needed to complete the transmission of the packet between input i and output j, $R_{ij} = 0, 1, 2 \cdots$.

Given a state Y, define the active VOQs whose remaining transmission time is larger than zero as:

$$\mathbf{B} = \{(i,j) | S_{ij} = 1,\ R_{ij} > 0\}. \tag{8}$$

Any crosspoint in $\mathbf{B}$ will stay in the schedule since the packet transmission is not finished yet. The crosspoints in $\mathbf{S} \cap \overline{\mathbf{B}}$ have finished a packet transmission. They can be removed from the schedule if they are picked by $\mathbf{H}(n)$.

According to the scheduling algorithm, VOQs can become active only when none of its neighbors were in the schedule of the previous time slot. So, the neighbors of $\mathbf{S}(n-1)$ are blocked, and will remain inactive.

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 $(i,j) \in \mathbf{B}$, $S'_{ij} = 1$, $L'_{ij} = L_{ij}$, and $R'_{ij} = R_{ij} - 1$.*
2) *At most one crosspoint is in $\mathbf{S} \cap \mathbf{S}'$, and $\mathbf{S} \cup \mathbf{S}' \in \mathcal{S}$.*

See Appendix A for the proof of Lemma 1.

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:*

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

where $a(\boldsymbol{H})$ is the probability that $\boldsymbol{H}$ is selected (which is $1/N^2$), and $\mathbf{S} \triangle \mathbf{S}' = (\mathbf{S} \cap \overline{\mathbf{S}'}) \cup (\overline{\mathbf{S}} \cap \mathbf{S}')$.

See Appendix B for the proof of Lemma 2.

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

$$\pi(\mathbf{Y}) = \frac{1}{\mathcal{Z}} \prod_{(i,j) \in \mathbf{S}} \frac{p_{ij}}{\overline{p}_{ij}} P_{ij}(L_{ij}), \tag{10}$$

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

$$\mathcal{Z} = \sum_{\mathbf{Y} \in \mathcal{Y}} \prod_{(i,j) \in \mathbf{S}} \frac{p_{ij}}{\overline{p}_{ij}} P_{ij}(L_{ij}). \tag{11}$$

Proof: Define the probability:

$$\begin{aligned} Q(\mathbf{Y}', \mathbf{Y}) \quad &= \sum_{\mathbf{H}:\mathbf{S}\triangle\mathbf{S}'\in\mathbf{H}} a(\mathbf{H}) \prod_{(k,l)\in\overline{\mathbf{S}}\cap\mathbf{S}'} \overline{p}_{kl} \\ &\bullet \prod_{(i,j)\in\mathbf{S}\cap\overline{\mathbf{S}}'} p_{ij} \times P_{ij}(L_{ij}) \\ &\bullet \prod_{(u,v)\in(\mathbf{S}\cap\overline{\mathbf{B}})\cap(\mathbf{S}'\cap\overline{\mathbf{B}})\cap\mathbf{H}} p_{uv} \times P_{uv}(L_{uv}) \\ &\bullet \prod_{(x,y)\in\mathbf{H}\cap\overline{\mathbf{S}\cup\mathbf{S}'}\cap\overline{\mathcal{N}(\mathbf{S}\cup\mathbf{S}')}} \overline{p}_{xy}. \end{aligned} \quad (12)$$

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

$$\pi(\mathbf{Y})P(\mathbf{Y}, \mathbf{Y}') = \pi(\mathbf{Y}')Q(\mathbf{Y}', \mathbf{Y}). \quad (13)$$

Therefore,

$$\begin{aligned} \sum_{\mathbf{Y}} \pi(\mathbf{Y}) \cdot P(\mathbf{Y}, \mathbf{Y}') &= \sum_{\mathbf{Y}} \pi(\mathbf{Y}') \cdot Q(\mathbf{Y}', \mathbf{Y}) \\ = \pi(\mathbf{Y}') \cdot \sum_{\mathbf{Y}} Q(\mathbf{Y}', \mathbf{Y}) &= \pi(\mathbf{Y}') \end{aligned} \quad (14)$$

The probability of state $\mathbf{Y}'$ is invariant over time, which means the distribution shown in Eq. (10) is stationary. ■

As we can see, the stationary distribution is a function of the probability p_{ij}, which is a concave function of the queue size. Q_{ij} is changing with time, so is the weight. The Markov chain is time-inhomogeneous. But as proved in [15], if the weight function $f(\cdot)$ is a strictly increasing concave function, and it grows slower that $\log(\cdot)$, the behavior of this algorithm will be very close to one with fixed weight in its stationarity if the arriving traffic satisfies SLLN. The probability distribution of the time-inhomogeneous Markov chain will then converge to a homogeneous chain.

E. 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:

$$\mathbf{S}^*(n) = \arg\max_{\mathbf{S}\in\mathcal{S}} \sum_{(i,j)\in\mathbf{S}} w_{ij}(n). \quad (15)$$

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

$$W(\mathbf{Y}) = W(\mathbf{S}) = \sum_{(i,j)\in\mathbf{S}} S_{ij}(n)w_{ij}(n). \quad (16)$$

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 [16].

Lemma 3. *For a scheduling algorithm, if given any ϵ and δ such that $0 \leq \epsilon$, $\delta < 1$, 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:*

$$\sum_{(i,j)\in\mathbf{S}(n)} w_{ij}(n) \geq (1 - \epsilon) \sum_{(k,l)\in\mathbf{S}^*(n)} w_{kl}(n), \quad (17)$$

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

We will prove the system stability using Lemma 3. Before the proof of Theorem 2, we first have to prove the lemma:

Lemma 4. *Let $W(\cdot)$ be the weight function and $W^*(\mathbf{S})$ the maximum weight. Define the set:*

$$\mathcal{K} = \{\mathbf{Y} \in \mathcal{Y} : W(\mathbf{Y}) = W(\mathbf{S}) \leq (1 - \epsilon)W^*(\mathbf{S})\}. \quad (18)$$

Then, we have:

$$\pi(\mathcal{K}) \leq \frac{\log|\mathcal{Y}| + |G_{min}|}{\epsilon W^*(\mathbf{S})}, \quad (19)$$

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.

See Appendix C for the proof of Lemma 4.

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:

$$W^*(\mathbf{S}) > \frac{\log|\mathcal{Y}| + |G_{min}|}{\epsilon\delta} > \frac{\log|\mathcal{Y}| + |G_{min}|}{\epsilon\delta}. \quad (20)$$

So, for any $\epsilon, \delta > 0$, there exists a $B > 0$ such that whenever $||\mathbf{Q}(n)|| > B$, Eq. (20) holds and then $\pi(\mathcal{K}) < \delta$. Hence the scheduling algorithm can stabilize the system according to Lemma 3. ■

F. Distributed Implementation

In the previous section, we presented a randomized scheduling algorithm with a complexity of $O(1)$. The algorithm requires a centralized scheduler to generate a schedule at every time slot. From the perspective of implementation, a distributed algorithm is more desirable. For example, in a 100Gbps line rate switch, a slot time is only about 5ns. A centralized scheduler may spend too much time collecting the information, and therefore introduce too much overhead. In this section, we will show the algorithm can be implemented in a distributed way, with only modest information exchange.

Each input scheduler has to keep track of the schedule of the previous time slot. For example, input i has to remember for which j was $S_{ij}(n-1) = 1$. At the beginning of time n, a new schedule $\mathbf{H}(n)$ has to be generated. The schedule $\mathbf{H}(n)$ can be pre-determined, for example, the crosspoint (i, j) which satisfies the condition $(i-1)*N+j-1 = (n \bmod N^2)$ is in the schedule $\mathbf{H}(n)$ at time n. After $\mathbf{H}(n)$ is generated, each input scheduler has to update its schedule based on the algorithm below.

Scheduling Algorithm at an Input Scheduler

```
At input $i$, if $(i,j)$ is in $\mathbf{H(n)}$ so that $H_{ij}(n)=1$.
∘ If $S_{ij}(n-1)=1$:
    - If the transmission between input $i$ and output $j$ is finished:
        (a) $S_{ij}(n)=1$ with probability $p_{ij}$;
        (b) $S_{ij}(n)=0$ with probability $\overline{p}_{ij}=1-p_{ij}$.
    - Else:
        (c) $S_{ij}(n)=S_{ij}(n-1)=1$.
∘ Else, if there exists a $(k,l)\in\mathcal{N}(i,j)$, $S_{kl}(n-1)$ = 1:
    (d) $S_{ij}(n)=S_{ij}(n-1)=0$.
∘ Else, if $\forall(k,l)\in\mathcal{N}(i,j)$, $S_{kl}(n-1)=0$:
    (e) $S_{ij}(n)=1$ with probability $p_{ij}$;
    (f) $S_{ij}(n)=0$ with probability $\overline{p}_{ij}=1-p_{ij}$.
```

From the algorithm, we can see that when the crosspoint (i,j) is selected by $\mathbf{H(n)}$ and $(i,j)\notin\mathbf{S}(n-1)$, the input i has to know the schedules of all crosspoints in $\mathcal{N}(i,j)=\{(i,j')\text{ or }(i',j)|\forall j'\neq j,\forall i'\neq i\}$ to make a scheduling decision. However, input i only has the information of the crosspoints in $\{(i,j')|\forall j'\}$, and does not know the schedules of the crosspoints in $\{(i',j)|\forall i'\neq i\}$. But this information is known to output j, because if their exists one i' such that $S_{i'j}(n-1)=1$, output j would receive a packet from input i' at time $n-1$. So, if output j can send the information regarding whether it was busy or not at time $n-1$, which is one-bit information, to input i, input i would have all the information it needs to make a decision.

Therefore, at the beginning of time n, if $(i,j)\in\mathbf{H}(n)$, output j has to send a one-bit data to input i. This can be done by using a low-rate channel. For example, when the data rate of each linecard is $100Gbps$ and each cell has a size of 64 Bytes, a channel with a data rate of $\frac{100Gbps}{64\times8}\approx0.2Gbps$ is sufficient for the one-bit message passing. This can be implemented using an *out-of-band signaling* channel, e.g., an Ethernet line.

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 our algorithm with two algorithms proposed in [8]: *Packet MWM* (P-MWM) and *Packet Incremental MWM* (PI-MWM).

A. P-MWM and PI-MWM

The packet MWM scheduling algorithm [3] 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 algorithm 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.

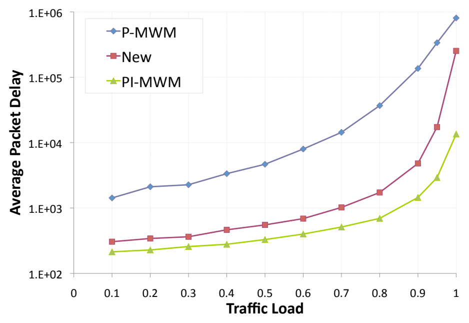

Fig. 2. N=16, uniform traffic, uniform packet size distribution

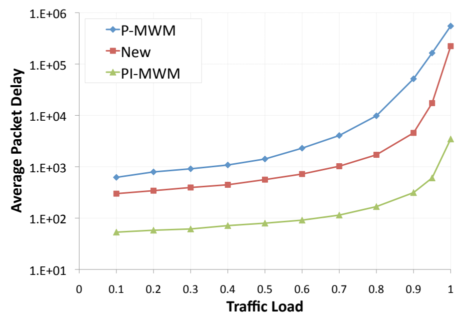

Fig. 3. N=16, uniform traffic, Pareto packet size distribution

- 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. As a result, 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 [8], 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 $p_a=0.5$.

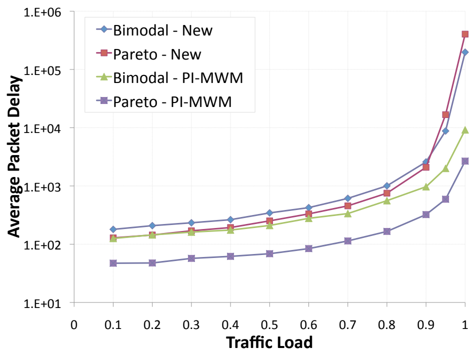

Fig. 4. Switch size $N = 16$, hot-spot traffic

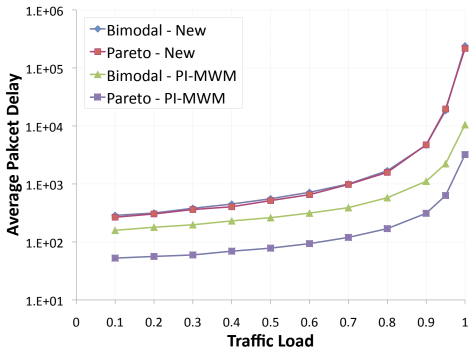

Fig. 5. Switch size $N = 16$, lin-diagonal traffic

3) Pareto(a, b). Packet sizes are distributed over $[a, b]$, following the truncated Pareto distribution:

$$P(l) = \frac{c}{l^{\beta}},\ l = \text{a, a+1, ... , b}, \quad (21)$$

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

We ran 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.

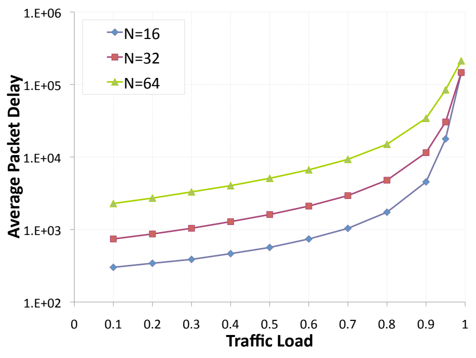

Fig. 6. Impact of switch size, lin-diagonal traffic

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 Pareto packet 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 can be seen in Fig. 5, the delay for bimodal and exponential distributions are almost the same.

D. Impact of Switch Size

In this section, we will study the impact of switch size on the delay performance. Generally, for input-queued switches, the average delay increases linearly with the switch size. Fig. 6 shows the delay performance of switches with different sizes under lin-diagonal traffic, with Bimodal packet size distribution. We can see that in common with other scheduling algorithms, the delay increases almost linearly with the switch size.

IV. Conclusion

In this paper, we propose a $O(1)$ complexity packet-mode scheduling algorithm for an input-queued switch. We prove that it can achieve 100% throughput under any admissible Bernoulli i.i.d. traffic. We then show that the algorithm can be implemented in a distributed way with a low rate control channel. Our simulation results show that it can provide very good delay performance under different traffic arrivals. The algorithm is therefore a good candidate for future large-scale high-speed switching systems.

References

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

[2] Y. Li, S. S. Panwar, and H. J. Chao, "Exhaustive Service Matching Algorithms for Input Queued Switches," in *Proc. of IEEE HPSR*, April 2004.

[3] L. Tassiulas 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.

[4] N. Mckeown, "The iSLIP Scheduling Algorithm for Input-Queued Switches," *IEEE/ACM Transactions on Networking*, vol. 7, pp. 188–201, April 1999.

[5] N. Mckeown, "Scheduling Algorithms for Input-queued Cell Switches," *Ph.D. Thesis, University of California at Berkeley*, 1995.

[6] Y. Li, S. Panwar, and H. J. Chao, "On the Performance of a Dual Round-Robin Switch," in *Proc. of IEEE INFOCOM*, April 2001.

[7] L. Tassiulas, "Linear complexity algorithms for maximum throughput in radio networks and input queued switches," in *Proc. of IEEE INFOCOM*, 1998.

[8] 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.

[9] Y. Ganjali, A. Keshavarzian, and D. Shah, "Input Queued Switches: Cell Switching vs. Packet Switching," in *Proc. of IEEE INFOCOM*, 2003.

[10] 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 HPSR*, June 2010.

[11] S. Rajagopalan and D. Shah, "Distributed Algorithm and Reversible Networks," in *Proc. of CISS*, March 2008.

[12] S. Rajagopalan, D. Shah, and J. Shin, "Network adiabatic theorem: an efficient randomized protocol for contention resolution," in *Proc. of ACM SIGMETRICS*, 2009.

[13] L. Jiang and J. Walrand, "A Distributed Algorithm for Optimal Throughput and Fairness in Wireless Networks with a General Interference Model," *IEEE/ACM Transactions on Networking*, vol. 18, pp. 960–972, June 2010.

[14] J. Ni and R. Srikant, "Q-CSMA: Queue-Length Based CSMA/CA Algorithms for Achieving Maximum Throughput and Low Delay in Wireless Networks," in *Proc. of IEEE INFOCOM Mini-Conference*, March 2010.

[15] J. Ghaderi and R. Srikant, "On the Design of Efficient CSMA Algorithms for Wireless Networks," *to be submitted*, 2010.

[16] 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.

APPENDIX

A. Proof of Lemma 1

Proof: (Necessity) For packet-mode switching, once we start a packet transmission, the input and output have to keep serving this packet until the transmission is completed. Therefore, for those active VOQs whose remaining time for the current packet transmission is not 0, they will stay in the schedule, and the remaining time will be reduced by one since one cell will be transmitted in current time slot. So condition 1) is necessary.

According to the scheduling algorithm, S_{ij} can change its value only when $(i,j) \in \mathbf{H}(n)$. There is only one crosspoint in $\mathbf{H}(n)$, so only one crosspoint can be added to or removed from $\mathbf{S}$. Thus, there is at most one crosspoint in $(\mathbf{S} \cap \overline{\mathbf{S}'}) \cup (\overline{\mathbf{S}} \cap \mathbf{S}')$.

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 no neighbors were in the schedule of 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 $(i,j) \in \overline{\mathbf{S}} \cap \mathbf{S}'$, $\forall (k,l) \in \mathcal{N}(i,j)$, $(k,l) \notin \mathbf{S}'$ as $\mathbf{S}'$ itself is also a feasible schedule. (k,l) cannot be in $\mathbf{S}$ as (i,j) changes its schedule decision from 0 to 1. If (k,l) is in $\mathbf{S}$, (i,j) cannot 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 a feasible schedule.

(Sufficiency) There is at most on crosspoint in $(\mathbf{S} \cap \overline{\mathbf{S}'}) \cup (\overline{\mathbf{S}} \cap \mathbf{S}')$. Thus there exists at least one $\mathbf{H}$ such that $(\mathbf{S} \cap \overline{\mathbf{S}'}) \cup (\overline{\mathbf{S}} \cap \mathbf{S}') \in \mathbf{H}$. When these $\mathbf{H}$ are selected by the random algorithm, 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. ■

B. Proof of Lemma 2

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 scheduling decision 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 scheduling decision from 0 to 1, which happens with probability p_{kl}. The length of the head-of-line packet in VOQ_{kl} is 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 $\mathbf{H}$ it decides to keep its schedule, which occurs with probability p_{uv}. The length of the 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 previous time slot, and though selected by $\mathbf{H}$ it decides to keep its schedule, which occurs with probability $\overline{p}_{xy}$. Since $\mathbf{H}$ 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. (9) for the proof of the stationary distribution in the following.

Since $\mathbf{H}$ is a feasible schedule, the four elements above can make their scheduling decisions independently. Therefore, we can multiply the probabilities of all the four categories above, which leads to the transition probability given by Eq. (9). ■

C. Proof of Lemma 4

Lemma 5. *Suppose that $T(\cdot)$ is a function defined on a set $\mathcal{Y}$. For any probability distribution μ on $\mathcal{Y}$, define the function:*

$$F(\mu, T(\mathbf{Y})) = E_{\mu}[T(\mathbf{Y})] + H(\mu), \tag{22}$$

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:

$$\mu^*(\mathbf{Y}) = \frac{1}{Z} \exp(T(\mathbf{Y})), \tag{23}$$

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

Proof: For any probability distribution μ, we have:

$$\begin{aligned}
& F(\mu, T(\mathbf{Y})) \\
= \;& E_\mu[T(\mathbf{Y})] + H(\mu) \\
= \;& \sum_{\mathbf{Y}\in\mathcal{Y}} \mu(\mathbf{Y})T(\mathbf{Y}) - \sum_{\mathbf{Y}\in\mathcal{Y}} \mu(\mathbf{Y}) \log \mu(\mathbf{Y}) \\
= \;& \sum_{\mathbf{Y}\in\mathcal{Y}} \mu(\mathbf{Y})(\log \mu^*(\mathbf{Y}) + \log Z) - \sum_{\mathbf{Y}\in\mathcal{Y}} \mu(\mathbf{Y}) \log \mu(\mathbf{Y}) \\
= \;& \sum_{\mathbf{Y}\in\mathcal{Y}} \mu(\mathbf{Y}) \log Z + \sum_{\mathbf{Y}\in\mathcal{Y}} \mu(\mathbf{Y}) \log \frac{\mu^*(\mathbf{Y})}{\mu(\mathbf{Y})} \\
\leq \;& \log Z \sum_{\mathbf{Y}\in\mathcal{Y}} \mu(\mathbf{Y}) + \log \Big(\sum_{\mathbf{Y}\in\mathcal{Y}} \mu(\mathbf{Y}) \frac{\mu^*(\mathbf{Y})}{\mu(\mathbf{Y})} \Big) \\
= \;& \log Z,
\end{aligned} \tag{24}$$

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

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

$$F(\mu, 0) = H(\mu) \leq \log Z = \log |\mathcal{Y}|. \tag{25}$$

Then we can prove Lemma 4:

Proof: As shown in Eq. (10), for a state $\mathbf{Y} = (\mathbf{S}, \mathbf{L}, \mathbf{R})$, its stationary distribution is:

$$\pi(\mathbf{Y}) = \frac{1}{\mathcal{Z}} \prod_{(i,j)\in\mathbf{S}} e^{(w_{ij}(n))} P_{ij}(L_{ij}) = \frac{1}{\mathcal{Z}} e^{W(\mathbf{S})} \prod_{(i,j)\in\mathbf{S}} P_{ij}(L_{ij}). \tag{26}$$

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

$$G(\mathbf{Y}) = \ln \prod_{(i,j)\in\mathbf{S}} P_{ij}(L_{ij}) \tag{27}$$

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. (27). Let G_{min} represent $\min_{\mathbf{Y}\in\mathcal{Y}} G(\mathbf{Y})$. Then $|G_{min}| \geq |G(\mathbf{Y})|$. According to Lemma 5, 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:

$$\begin{aligned}
& F(\pi', W(\mathbf{Y}) + G(\mathbf{Y})) \\
= \;& E_{\pi'}[W(\mathbf{Y}) + G(\mathbf{Y})] + H(\pi') \\
= \;& W^*(\mathbf{S}) + G(\mathbf{Y}^*) + H(\pi') \\
\leq \;& F(\pi, W(\mathbf{S}) + G(\mathbf{Y})) = E_\pi[W(\mathbf{Y}) + G(\mathbf{Y})] + H(\pi) \\
= \;& \sum_{\mathbf{Y}\in\mathcal{K}} \pi(\mathbf{Y})\mathbf{W}(\mathbf{Y}) + \sum_{\mathbf{Y}\in\overline{\mathcal{K}}} \pi(\mathbf{Y})\mathbf{W}(\mathbf{Y}) + E_\pi[G(\mathbf{Y})] + H(\pi) \\
\leq \;& \pi(\mathcal{K})(1-\epsilon)W^*(\mathbf{S}) + (1 - \pi(\mathcal{K}))W^*(\mathbf{S}) + G_{max} + H(\pi) \\
\leq \;& W^*(\mathbf{S})(1 - \epsilon\pi(\mathcal{K})) + G_{max} + H(\pi)
\end{aligned} \tag{28}$$

So,

$$\begin{aligned}
& W^*(\mathbf{S}) + G(\mathbf{Y}^*) + H(\pi') \\
\leq \;& W^*(\mathbf{S})(1 - \epsilon\pi(\mathcal{K})) + G_{max} + H(\pi) \\
\epsilon\pi(\mathcal{K})W^*(\mathbf{S}) \leq \;& H(\pi) - H(\pi') + G_{max} - G(\mathbf{Y}^*) \\
\leq \;& H(\pi) + |G_{min}| \leq \log |\mathcal{Y}| + |G_{min}| \\
\pi(\mathcal{K}) \leq \;& \frac{\log |\mathcal{Y}| + |G_{min}|}{\epsilon W^*(\mathbf{S})}
\end{aligned} \tag{29}$$

■