Feedback-Based Scheduling for Load-Balanced Two-Stage Switches

Bing Hu, *Student Member, IEEE*, and Kwan L. Yeung, *Senior Member, IEEE*

***Abstract*—A framework for designing feedback-based scheduling algorithms is proposed for elegantly solving the notorious packet missequencing problem of a load-balanced switch. Unlike existing approaches, we show that the efforts made in load balancing and keeping packets in order can complement each other. Specifically, at each middle-stage port between the two switch fabrics of a load-balanced switch, only a single-packet buffer for each virtual output queueing (VOQ) is required. Although packets belonging to the same flow pass through different middle-stage VOQs, the delays they experience at different middle-stage ports will be identical. This is made possible by properly selecting and coordinating the two sequences of switch configurations to form a joint sequence with both *staggered symmetry property* and *in-order packet delivery property*. Based on the staggered symmetry property, an efficient feedback mechanism is designed to allow the right middle-stage port occupancy vector to be delivered to the right input port at the right time. As a result, the performance of load balancing as well as the switch throughput is significantly improved. We further extend this feedback mechanism to support the multicabinet implementation of a load-balanced switch, where the propagation delay between switch linecards and switch fabrics is nonnegligible. As compared to the existing load-balanced switch architectures and scheduling algorithms, our solutions impose a modest requirement on switch hardware, but consistently yield better delay-throughput performance. Last but not least, some extensions and refinements are made to address the scalability, implementation, and fairness issues of our solutions.**

***Index Terms*—Load-balanced switch, feedback-based switch, two-stage switch.**

I. Introduction

DUE to the wide use of wavelength division multiplexing (WDM) technology in fiber, the transmission capacity increases sharply, while the processing capacity of current commercial switches/routers increases slowly. This speed mismatch makes the need for building high-speed routers urgent [1]. A major bottleneck of high-speed router design is its switch architecture, which is concerned with how packets are moved from one linecard to another. For switches that employ output queuing, each output can receive up to the maximum of N packets in each time slot, where N is switch size. The switch fabric and output ports must operate at N times the link rate. This makes output-queued switches difficult to scale.

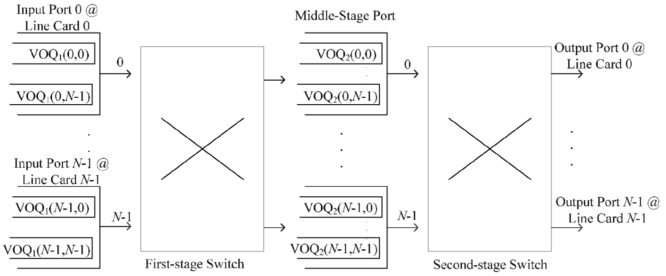

Fig. 1. A load-balanced two-stage switch.

For an input-queued switch, each input can send at most one packet, and each output can receive at most one packet in every time slot. The switch fabric only needs to run at the same speed as each input link. Input-queued switches are thus more suitable for high-speed routers. However, input-queued switches suffer from the head-of-line (HOL) blocking. To eliminate the HOL blocking, virtual output queuing (VOQ) is proposed [2], where each input port maintains a separate queue for each output. A centralized scheduler is needed to maximize the throughput of a VOQ switch. It is shown that for any admissible traffic patterns, 100% throughput can be achieved by maximum weight matching [3]. However, maximum weight matching algorithm has a high time complexity of $\mathrm{O}(N^3 \log N)$. Algorithms with lower computation overheads, notably PIM [4], iSLIP [5], [6] and DRRM [7], are then proposed. They share a common feature of trading more communication overheads for less computation overheads. However, the communication overheads scale up very quickly as the link speed and switch size increase.

Load-balanced switches have received a great deal of attention recently [8]–[19] because they are more scalable and can provide close to 100% throughput. A load-balanced switch consists of two stages of switch fabrics, as shown in Fig. 1. Each switch fabric is configured according to a predetermined and periodic sequence of switch configurations. This removes the need for a centralized scheduler. As a load-balanced switch provides multiple paths for packets belonging to the same flow to arrive at the same output port, packets may arrive out of order due to different middle-stage port delays experienced en route. Many efforts [9]–[19] are then made to address this notorious packet missequencing problem (reviewed in Section II). It is not difficult to see that higher switch throughput is usually at the cost of poorer delay performance. This is because throughput is improved by better load balancing, but better load balancing tends to deteriorate the packet mis-sequencing problem.

In this paper, we show that the efforts made in load balancing and keeping packets in order can complement each other in improving both delay and throughput performance of the switch. We adopt a simple switch architecture where each middle-stage

Manuscript received November 28, 2008; revised June 12, 2009 and August 25, 2009; approved by IEEE/ACM Transaction On Networking Editor C. S. Chang. First published December 28, 2009; current version published August 18, 2010. This work was supported in part by the Cisco Research Center.

The authors are with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong (e-mail: binghu@eee.hku.hk; kyeung@eee.hku.hk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2009.2037318

port between the two stages of switch fabrics only has a single-packet buffer for each VOQ. Although packets belonging to the same flow will pass through different middle-stage VOQs, the delays they experience at different middle-stage ports will be identical. This is made possible by properly selecting and coordinating the two sequences of switch configurations to form a joint sequence with both staggered symmetry property and in-order packet delivery property. Based on the staggered symmetry property, an efficient feedback mechanism is designed to allow the right middle-stage port occupancy vector to be delivered to the right input port at the right time. Accordingly, the performance of load balancing as well as switch throughput is significantly improved.

The rest of the paper is organized as follows. In the next section, we review the existing work for solving the packet missequencing problem of load-balanced two-stage switches. In Section III, our proposed feedback-based scheduling framework is introduced. In Section IV, we extend it to support the multi-cabinet implementation. The delay and throughput performance of our proposed solutions is compared to other existing algorithms in Section V by simulations, and the stability analysis is carried out in Section VI. In Section VII, some possible extensions and refinements of our proposed solution are discussed. We then compare our work with some closely related work in Section VIII. Finally, we conclude the paper in Section IX.

II. Packet Missequencing in Load-Balanced Switches

A load-balanced two-stage switch architecture consists of two stages of switch fabrics. Each fabric is configured according to a predetermined and periodic sequence of switch configurations, with the only requirement that each input connects to each output exactly once in the sequence. The two fabrics can use different sequences. There are many ways to generate such a sequence, e.g., a sequence can be constructed by cyclic shifting the set of input/output connections used in each time slot, such that at time slot t, input i (for $i = 0, 1, 2, \ldots, N-1$) is connected to output j, where j is given by

$$j = (i + t) \bmod N. \tag{1}$$

Consider a generic two-stage load-balanced switch architecture shown in Fig. 1. We use $\mathrm{VOQ}_1(i,k)$ to represent the VOQ at input port i with packets destined for output k, and $\mathrm{VOQ}_2(j,k)$ to denote the VOQ at middle-stage port j with packets destined for output k. We define $\mathrm{flow}(i,k)$ as packets arriving at input i and destined for output k. Packets from $\mathrm{flow}(i,k)$ are buffered at $\mathrm{VOQ}_1(i,k)$. Packets (from different inputs) destined for output k are buffered at $\mathrm{VOQ}_2(j,k)$ for $j = 0, 1, \ldots, N-1$. Aiming to convert the incoming nonuniform traffic to uniform, the first-stage switch fabric spreads packets evenly over all middle-stage ports. Then, the second-stage switch fabric delivers the packets from middle-stage ports to their respective outputs. From the above, we can see that in each time slot, there are two switch configurations, one at each fabric. We call them a *joint* configuration. The sequence of N joint configurations forms a *joint sequence*. Three possible joint sequences are shown in Fig. 2. In Fig. 2, each port is abstracted as a circle, and the configurations used by the first-stage switch fabric are shown by lines from the left side to the center. As an example, in Fig. 2(a), the sequence of configurations used by the first-stage switch fabric is constructed based on the cyclic shifting in (1). It is important to point out that all three joint sequences in Fig. 2 meet the basic requirement of a load-balanced two-stage switch, but they have different properties, namely, in-order packet delivery and staggered symmetry. These two properties will be discussed in detail in Section III, which form the basis of our feedback-based switch design.

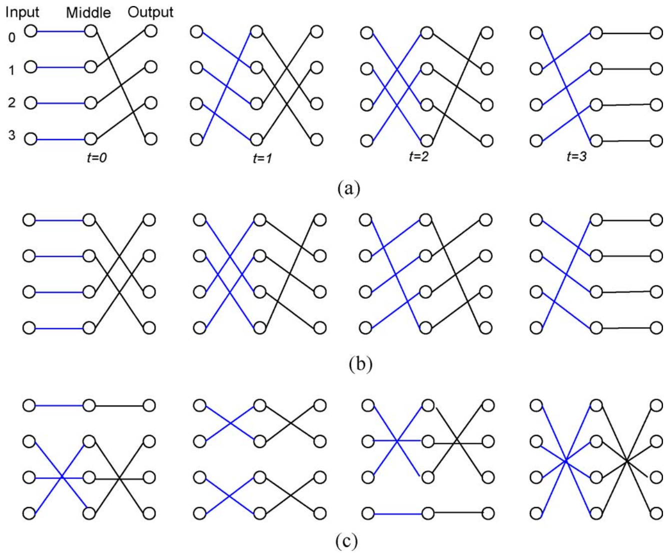

Fig. 2. Some joint sequences for a 4 × 4 two-stage switch. (a) Staggered symmetry + in-order packet delivery. (b) Staggered symmetry only. (c) In-order packet delivery only.

Due to the two-stage nature, $\mathrm{flow}(i,k)$ packets may arrive at output k via different middle-stage $\mathrm{VOQ}_2(j,k)$'s (for $j = 0, 1, \ldots, N-1$) and thus may experience different amounts of middle-stage port delay. This leads to the problem of packet missequencing. Two main approaches can be followed to solve this problem, using resequencing buffers at outputs or preventing packets from becoming missequenced in the first place.

A. Using Resequencing Buffers

When out-of-order packets arrive at an output port, they are stored in the resequencing buffer (not shown in Fig. 1), waiting to be read out and written onto the output link in the correct order. To this end, each packet header should have a sequence number field (or timestamp), which is added to the packet at an input port. With the original two-stage switch architecture [8], packets can be missequenced by an arbitrary amount, thus a finite resequencing buffer is not possible. Efforts are made to bound the delay at additional costs, such as N writes to memory in one time slot [9] and a 3-D resequencing buffer [10]. Recently, a three-stage load-balancing switch [11] is also proposed. Although the switch is proved to be stable, this design requires additional hardware as well as global information exchange for buffer reservation. The high implementation complexity may defeat the original purpose of using a load-balanced switch.

B. Preventing Packets From Becoming Missequenced

Instead of reordering packets at each output, we can prevent packets from becoming missequenced in the first place

[12]–[19]. This not only removes the resequencing buffers, but also the corresponding resequencing delay. The majority work along this direction [13]–[16] adopts the notion of "frame." For an $N \times N$ switch, a frame consists of N packets belonging to the same flow. At each input port, incoming packets join their respective VOQs. If the size of a VOQ is larger than N packets, the flow is said to have a full frame of packets. With the Uniform Frame Spreading (UFS) algorithm [13], an input port is allowed to send only from flows/VOQs with at least a frame of packets. Once the sending starts, N packets from the selected flow will be sent in the next N slots, where each packet arrives at a distinct middle-stage port.

A downside of the UFS algorithm is that when traffic load is light, it takes time to form a full frame of packets, thus the delay performance suffers. To cut down the delay, Full Ordered Frames First (FOFF) [14] is proposed. FOFF allows the sending of partial frames, which is at the cost of controlled packet missequencing. Nevertheless, resequencing buffer at each output is required.

Padded Frame (PF) algorithm [15] also improves the delay performance of UFS, and without resequencing buffer. The idea is that when no full frames are available for sending, a partial frame can be sent as a "faked" full frame by padding the partial frame with dummy packets. Contention-and-Reservation (CR) algorithm [16] can further improve the performance of PF by supporting two modes of frame transmission: contention and reservation. As long as an input i has a full frame of packets when i connects to middle port 0, i enters the reservation mode, and the transmission in the next N slots is governed by UFS. Otherwise, input i enters the contention mode, where the packet sent in each slot is selected using a round-robin scheduler, and must be acknowledged at the end of each time slot. A packet is removed from the input VOQ only if a positive ACK is received.

CR algorithm requires dedicated ACK from each middle-stage port in each time slot. The feedback path construction is not discussed in [16]. To this end, the Mailbox switch [17] is designed with a rather efficient feedback path. The feedback path is constructed by adopting the joint sequence of switch configurations in Fig. 2(c), where input i and output i are always connected to the same middle-stage port. When a packet arrives at a middle-stage port (from, say, input i), the middle-stage port calculates its departure time based on its location in the VOQ. Then, the departure time is sent to the connected output port i using the second switch fabric. As input i and output i reside on the same switch linecard, output i can relay the departure time to input i at negligible cost. A feedback path for reporting middle-stage packet departure time is thus created. Based on the received packet departure time, the next packet in the flow will be dispatched and inserted in a middle-stage VOQ if it will depart no earlier than the previous packet of the same flow. Although the packet order is maintained by Mailbox switch without the frame notion, the overall switch throughput is limited. By modifying each middle-stage $\mathrm{VOQ}_2(j,k)$ of the Mailbox switch to just accommodate a single packet, the in-order packet delivery property of the joint sequence in Fig. 2(c) is discovered by Lee [19]. Our work in this paper, developed independently, is most closely related to [19]. Please refer to a more focused comparison in Section VIII-A.

In [18], a distributed and iterative scheduling algorithm, Concurrent Matching Switch (CMS), is introduced. Despite the fixed uniform identical mesh in both stages of switch fabrics, its logical configurations are the same as the joint sequence in Fig. 2(c). For every arriving packet, input port sends a request to the current (logically) connected middle-stage port. Each middle port records the requests in its matrix $\{\mathrm{r}_{i,j}\}$, where $\mathrm{r}_{i,j}$ denotes the number of requests from $\mathrm{flow}(i,j)$. Every N slots, each middle-stage port concurrently and independently finds a matching based on its own $\{\mathrm{r}_{i,j}\}$. In the following N slots, the packets matched are transmitted to the middle-stage ports and are immediately forwarded to the output ports. Since the packets selected in each slot traverse the two switches in parallel and without conflicts, there is no out-of-order problem. However, the packet delay can be quite large, where the best case is three N time slots when a parallel optical mesh is used. Nevertheless, the delay performance of [8] is on the order of $\mathrm{O}(N)$ if it is implemented using an R/N optics abstraction.

III. Feedback-Based Scheduling Framework

A. Observations and Motivations

The delay and throughput performance of a load-balanced two-stage switch hinges on how well the load-balancing and in-order packet delivery are implemented. Obviously, if the incoming traffic is well balanced by the first-stage switch, the throughput performance will be improved as the second-stage switch can maximize the number of packets sent in each time slot. Consequently, the packet delay will also be reduced due to higher throughput.

But how to measure the load-balancing performance? Many scheduling algorithms (e.g., [10] and [12]) try to ensure all middle-stage VOQs have the same queue size. However, as far as the throughput performance is concerned, we only need to ensure each middle-stage $\mathrm{VOQ}_2(j,k)$ (in Fig. 1) does not suffer from either buffer underflow or overflow problem. A buffer underflow occurs if there are packets waiting in some input ports for a particular output k, but $\mathrm{VOQ}_2(j,k)$ is empty at the time that middle-stage port j is connected to output k, yielding an idle transmission slot on the second-stage switch. On the other hand, buffer overflow is equally undesirable as the overflowed packet is dropped, and the transmission slot in the first-stage switch is wasted. Indeed, as long as no buffer underflow and overflow at each $\mathrm{VOQ}_2(j,k)$ is ensured, the actual buffer size for each $\mathrm{VOQ}_2(j,k)$ has no impact on the throughput performance of the switch. Therefore, it may not be appropriate to increase the buffer size of $\mathrm{VOQ}_2(j,k)$ for boosting throughput performance.

In a load-balanced switch, the HOL packet in each middle-stage VOQ will experience an average delay of $N/2$ slots (due to the deterministic nature of the N configurations), and each additional packet in the line will experience an additional delay of N slots. To minimize delay, a small buffer size at each $\mathrm{VOQ}_2(j,k)$ is preferred.

In general, mechanisms for ensuring in-order packet delivery tend to penalize the packet delay performance more than throughput. If resequencing buffers are used for solving the missequencing problem, packets suffer from the additional resequencing delay. Since packet missequencing is due to packets of the same flow experiencing different delays at different middle-stage ports, a smaller buffer size at each $\mathrm{VOQ}_2(j,k)$ is favored because middle-stage packet delay can be reduced, and thus the missequencing problem can be eased. Consequently, a smaller resequencing buffer/delay is also possible. In fact, buffering a packet at an input port (instead of a middle-stage port) gives more flexibility in sending because an input can retry in the subsequent slots at different middle-stage ports (which may even have a shorter queue size).

If the frame notion is used for ensuring in-order packet delivery, the time required for forming a frame dominates the delay performance especially when the load is light. Besides, frame-based transmission tends to make the traffic to downstream switches more bursty, resulting in poor delay jitter performance. Although PD [15] and CR [16] improve the delay performance of UFS [13], the use of fake frames/packets undermines the load-balancing performance. In this paper, we are interested in designing a scheduling algorithm without using resequencing buffers for in-order packet delivery and without incurring the frame-based scheduling overheads.

From our observations above, we can see that a smaller buffer size at each $\mathrm{VOQ}_2(j,k)$ is preferred if we can ensure: 1) no underflow and overflow at each $\mathrm{VOQ}_2(j,k)$; and 2) no packet missequencing. The smallest buffer size at each $\mathrm{VOQ}_2(j,k)$ is 1. In the rest of the paper, we shall focus on using a single-packet buffer at each $\mathrm{VOQ}_2(j,k)$.

B. Designing Scalable Feedback Mechanism

Now the issue is how to ensure each single-packet-buffered $\mathrm{VOQ}_2(j,k)$ is free of either buffer overflow or underflow. If an input port knows the occupancy of its connected $\mathrm{VOQ}_2(j,k)$ *before* sending a packet to it, the buffer overflow problem can be easily solved. Then, do we have an efficient feedback mechanism for reporting the occupancy of $\mathrm{VOQ}_2(j,k)$ to input ports?

We propose a simple yet novel feedback mechanism based on a joint sequence with *staggered symmetry* property. A joint sequence of switch configurations has the *staggered symmetry* property if middle-stage port j is connected to output port k at time slot t, then at next slot $(t+1)$ input port k is connected to the same middle-stage port j. In essence, for each given sequence in the first-stage switch, the second-stage sequence (and thus the joint sequence) can be obtained directly from the property itself. In Fig. 2(a), the first stage sequence is constructed from (1) by cyclic shifting the set of connections used in each slot. Each configuration in the second stage is obtained from the staggered symmetry property. We can see that for every pair of *staggered* configurations, e.g., the second switch configuration at $t=0$ and the first switch configuration at $t=1$, they are mirror images of each other.

As each $\mathrm{VOQ}_2(j,k)$ only has a single packet buffer, a single bit is sufficient to denote its occupancy. For the N $\mathrm{VOQ}_2(j,k)$'s at middle-stage port j (for $k=0,\ldots,N-1$), their joint occupancy can be denoted by an N-bit occupancy vector. Since each pair of input k and output k reside on the same linecard, the occupancy vector at middle-stage port j can be *piggybacked* on the data packet sent to output k, which is then made available to input k at negligible cost. Due to the staggered symmetry property of the joint sequence used, input k will be connected to middle port j in the next time slot. This gives a very efficient feedback path, allowing the occupancy vector from the *right* middle-stage port to be delivered to the *right* input at the *right* time. In the next time slot, each input port scheduler will select a packet for sending based on the received occupancy vector. If the packet is properly selected, both buffer overflow and underflow at a middle-stage $\mathrm{VOQ}_2(j,k)$ can be avoided. (In Section III-E, three simple input port schedulers are designed.)

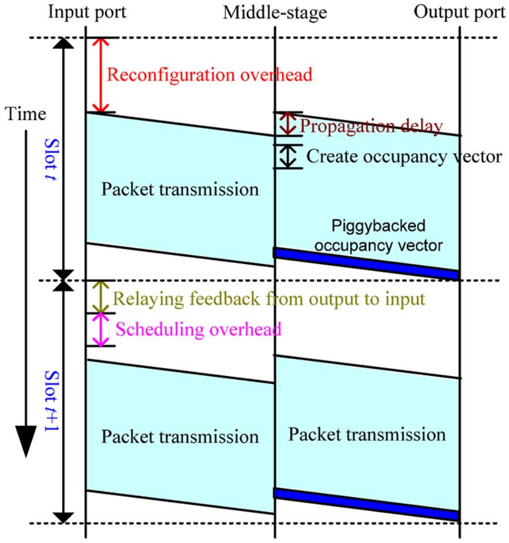

Fig. 3. Feedback operation based on joint sequences with staggered symmetry property.

The timing diagram in Fig. 3 summarizes the feedback operation, while assuming each switch reconfiguration involves certain overhead. We can see that switch reconfiguration takes place in parallel with relaying the occupancy vector from output k to input k and the execution of the scheduling algorithm. The occupancy vector is created by taking both packet arrival/departure in the current slot into account. In creating the vector, the occupancy bit of $\mathrm{VOQ}_2(j,k')$ is always set to 0 if middle port j will connect to output k' in the next slot. This is because the packet (if any) in $\mathrm{VOQ}_2(j,k')$ is guaranteed to be sent in the next time slot. Besides, when a buffered packet in $\mathrm{VOQ}_2(j,k')$ is being sent, $\mathrm{VOQ}_2(j,k')$ can receive another packet simultaneously. Due to parallel packet transmission in the two switch stages, a packet cannot be delivered from an input to an output in a single time slot, i.e., the minimum delay a packet experienced at a middle-stage port is one slot.

From Fig. 3, we can also see that the feedback operation requires accurate timing synchronization within a time slot. We notice that accurate synchronization of less than 10 ns is reported in [33], and a scheme to achieve 1-ns synchronization is proposed in [34]. Therefore, synchronization within a time slot of, say, 40 ns would not be a major issue.

Note that the joint sequence in Fig. 2(c) does not have the staggered symmetry property. If it is used for implementing feedback path (as in [17] and [19]), occupancy vector cannot be piggybacked onto data packet. Instead, a *dedicated* feedback packet must be sent from each middle-stage port to its connected output in each time slot. This incurs not only extra propagation delay for sending the feedback packet, but also extra packetization and synchronization overhead. As a result, the duration of a time slot in [17] and [19] would be much longer than that shown in Fig. 3. If the switch performance is studied using the number of time slots, the inefficiencies of using a "larger" time slot could be easily overlooked.

C. Solving Packet Missequencing Problem

If the load-balanced switch in Fig. 1 is configured by the joint sequence in Fig. 2(a), will we face the packet missequencing problem? We know that packet order will be preserved if *every* packet of a flow experiences the same amount of delay when passing through *any* middle-stage port. This is obviously true if middle-stage ports are bufferless, thereby every packet experiencing the same zero-slot delay. Will it still be true for the case of single-packet buffer per $\text{VOQ}_2(j,k)$?

Surprisingly, a closer examination at the joint sequence in Fig. 2(a) reveals that packets of the same flow do experience the same middle-stage port delay. Take flow(0,1) in Fig. 2(a) as an example. If a packet is sent (from input 0) to middle-stage port 0 at $t=0$, it will be buffered at VOQ_2(0,1) for two slots until VOQ_2(0,1) is connected to output 1 at $t=2$. If the next packet of the flow is sent to middle-stage port 1 at $t=1$, it will be buffered at VOQ_2(1,1) for, again, two slots until VOQ_2(1,1) is connected to output 1 at $t=3$.

In the following, we prove that this is true for each and every flow and for any switch size N. Consider the joint sequence in Fig. 2(a). The sequence used by the first stage switch is constructed from (1). The sequence used by the second stage switch is constructed according to the staggered symmetry property, which can be represented by (2). That is, at time t (for $0 \le t < N$), middle-stage port j is connected to output k, where k is given by

$$k = (j + N - 1 - t) \bmod N. \tag{2}$$

Statement 1: **(Anchor Output)**. Input i is always connected to output K, where $K = [(i+N-1) \bmod N]$, via one of the middle-stage ports.

Proof: At time t, input i is connected to output k via middle-stage port j. Substitute j from (1) into (2), we can express k in terms of i.

$$\begin{aligned} k &= [(i+t) \bmod N + N - 1 - t] \bmod N \\ &= (i + N - 1) \bmod N \\ &= K. \end{aligned}$$

We can see that K depends only on i. Thus, for a given input i, it is always connected to the same anchor output K. ■

Statement 2: (Deterministic Delay at Middle-stage Ports). Let K be the anchor output of input i. For every packet of flow(i,k), it experiences the same d slots delay in one of the middle-stage ports, where d is given by

$$d = \begin{cases} N, & \text{if } K = k \\ K - k, & \text{if } K > k \\ K + N - k, & \text{if } K < k. \end{cases} \tag{3}$$

Proof: Suppose at slot t, input i is connected to its anchor output K via middle-stage port j and a packet is sent to join $\text{VOQ}_2(j,k)$. From (2), middle port j is connected to each output in *descending* order of the output port number. Assume output port k, the destination of packet in $\text{VOQ}_2(j,k)$, is d ports away from the anchor output K (counted in descending order of port number). This packet will experience exactly d slots delay in $\text{VOQ}_2(j,k)$ and d is bounded by $[1,N]$. ■

Statement 3 (In-order Packet Delivery). In-order packet delivery is guaranteed if the joint sequence of configurations is constructed using (1) and (2).

Proof: Assume packets A and B of flow(i,k) join $\text{VOQ}_2(j_1,k)$ and $\text{VOQ}_2(j_2,k)$ at time t_A and t_B (where $t_\text{B} > t_\text{A}$), respectively. Let d_A and d_B be their respective delays experienced in VOQ_2. Missequencing occurs only if packet B reaches output k earlier than packet A, i.e., $t_\text{A} + d_\text{A} > t_\text{B} + d_\text{B}$. However, this will never happen because $t_\text{B} > t_\text{A}$ and $d_\text{A} = d_\text{B}$ from Statement 2. ■

It can be easily seen that the delay a packet experienced at a middle-stage port is bounded between $[1,N]$ slots, and the average middle-stage packet delay is merely $(N+1)/2$ slots for uniform traffic.

D. Alternative Joint Sequence Design

The total number of possible joint sequences for a load-balanced two-stage switch can be found by solving the classic Latin square problem [20], or $N![(N-1)!]^3M^2$, where M (≥ 1) is the number of reduced Latin squares. Among them, some joint sequences have the staggered symmetry property only, some have the in-order packet delivery property only, and some have both properties.

For instance, the joint sequence in Fig. 2(b) has the staggered symmetry property but cannot ensure in-order packet delivery. Consider packets from flow(0,1). Two different middle-stage delays will be experienced, two-slot via middle port 3 and four-slot via middle port 1. This causes out-of-order packet. On the other hand, the joint sequence in Fig. 2(c) can provide in-order packet delivery but lacks the staggered symmetry property. It is shown [21] that there are $[(N-1)!]^2$ joint sequences having both staggered symmetry and in-order packet delivery properties. However, our proposed feedback-based scheduling framework can be applied to any of them. For a specific traffic pattern, an optimal joint sequence can be found from this group for minimizing the delay performance. As far as this paper is concerned, we only focus on the joint sequence in Fig. 2(a).

E. Port-Based Scheduling Algorithms

Based on the received occupancy vector, each input port selects a packet for sending. Such an input port scheduler should be designed to avoid both buffer overflow and underflow at the connected middle-stage VOQ.

Suppose input i is connected to middle-stage port j at slot t, and its anchor output is K. Based on the N-bit occupancy vector received from middle-stage j in the previous slot $t-1$, we find candidate set S_j, i.e., the set of $\text{VOQ}_2(j,k)$ (for $k = 0,1,\ldots,N-1$) with 0-occupancy. Input i can only choose a HOL packet from a VOQ in S_j for sending. This avoids buffer overflow at $\text{VOQ}_2(j,h)$.

From Fig. 2(a), we can see that middle port j is connected to each output in descending order of the output port number. Therefore, we know *a priori* that in the next slot $t+1$, port j will be connected to output $K-1$ (wrapped around by N). If $\text{VOQ}_2(j,K-1)$ is empty and $\text{VOQ}_1(i,K-1)$ is not, we will face an underflow in $\text{VOQ}_2(j,K-1)$ at slot $t+1$. As such, the scheduling algorithm should always give the highest priority to schedule the HOL packet of $\text{VOQ}_1(i,K-1)$ at slot t.

With the above considerations in mind, we present three simple input port schedulers.

- *Round-Robin* (RR): If $\mathrm{VOQ}_1(i,h')$ is selected in the previous slot, then the next nonempty $\mathrm{VOQ}_1(i,h)$ is selected with $\mathrm{VOQ}_2(j,h) \in S_j$. **Comment:** RR gives fair access to each VOQ_1, and RR is amenable to hardware implementation [22].
- *Longest Queue First* (LQF): Among all the nonempty $\mathrm{VOQ}_1(i,h)$'s with $\mathrm{VOQ}_2(j,h) \in S_j$, the one with the longest queue size is selected. **Comment:** LQF is good for nonuniform traffic, but requires $O\,(N)$ comparisons. We can replace it by Quasi-LQF [23], a very efficient suboptimal LQF algorithm requiring only a single comparison per time slot.
- *Earliest Departure First* (EDF): Among all the nonempty $\mathrm{VOQ}_1(i,h)$'s with $\mathrm{VOQ}_2(j,h) \in S_j$, the one with the earliest departure time at the middle-stage port is selected. The departure time is calculated from (3). **Comment:** EDF should not be confused with the classic Earliest Deadline First. Our EDF aims at minimizing the chance of buffer overflow at each VOQ_2, which is achieved by always giving priority to the VOQ_1 with the minimum middle-stage delay to send first.

To give a scheduler more time to execute, batch scheduling [24], [25] can be used, where a single scheduling decision is made over a batch of time slots (instead of per slot). Packets arrived in the current batch of slots will be considered in the next batch. Indeed, the multicabinet implementation of the feedback-based switch in the next section belongs to this category.

IV. Multicabinet Implementation

To accommodate the growth of the Internet traffic, high-speed packet switches consist of a large number of linecards, resulting in larger physical space and power requirement. Consequently, a multicabinet implementation of packet switches is needed [26], where the distance between linecards and (central) switch fabrics can be tens of meters.

In a single-cabinet implementation, the propagation delay between linecards and switch fabrics is negligible. In a multicabinet implementation, due to the nonnegligible propagation delay, the requirement that occupancy vectors must arrive at input ports within a single time slot will significantly lower the switch efficiency. This is illustrated in Fig. 4. Since the occupancy vector needs to take the in-flight packet (in the first switch fabric) into account, it can only be generated when the packet (at least partly) arrives. A dedicated feedback packet is required as piggybacking occupancy vector onto data packet is not possible. Finally, an input port must wait for the occupancy vector to arrive before another packet can be scheduled for sending. From Fig. 4, we can see that the duration of a slot must be at least twice the propagation delay between linecards and the switch fabrics. However, in each slot, only a single packet can be sent. Since a switch fabric cannot be reconfigured while there are in-flight packets, the slot duration is (roughly) the duration that a switch configuration lasts.

To increase the switch efficiency, we can send multiple packets in a slot. The minimum duration of a slot is the round-trip propagation time between linecards and switch fabrics, or RTT seconds. Let the (maximum) number of packets that can be sent in each slot be x. The value of x depends on packet size (B bytes), RTT, and the line rate (R bps). Roughly, we have

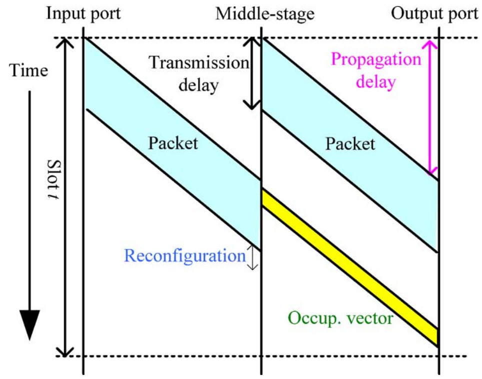

Fig. 4. Feedback operation with large propagation delay between linecards and switch fabrics.

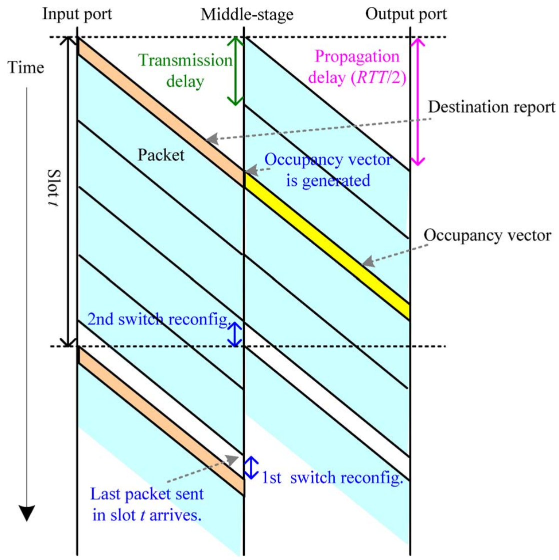

Fig. 5. Feedback operation in multicabinet implementation.

$$x = \lfloor RTT/\text{packet\_duration} \rfloor = \left\lfloor \frac{RTT \cdot R}{8B} \right\rfloor.$$

For a typical distance of 20 m between linecards and switch fabrics, the (minimum) slot duration is $RTT = 200$ ns. To transmit a packet of 100 bytes on a 20-Gbps line, 40 ns are required. Reserving some guard times for control, we can transmit $x = 4$ packets in a slot, as shown in Fig. 5.

However, can we still keep the in-order packet delivery and high-throughput properties of a single-cabinet implementation of the feedback switch? With the following modifications, the answer is yes. First of all, the buffer size at each middle-stage port $\mathrm{VOQ}_2(j,k)$ is increased to x to accommodate up to x packet arrivals in each time slot. The occupancy vector is expanded to $N \log x$ bits, as the size of each VOQ requires $\log x$ bits.

The feedback operation is also revamped. Refer to Fig. 5. Assume at time slot t, input port i connects to output k via middle stage port j. At the beginning of slot t, (based on the occupancy vector received in the previous slot) input i uses a local batch scheduler (to be detailed) to select up to x packets for sending. A special header (destination report) is appended to the first packet sent, which contains the destinations of the x packets (to be) sent in this slot. As each destination requires $\log N$ bits to denote, the destination report consists of $x \log N$ bits.

While input ports are sending packets to middle-stage ports, middle-stage ports are sending packets to output ports in parallel. When a middle-stage port (j) is connected to an output port (k), all backlogged packets in $\mathrm{VOQ}_2(j,k)$ (at most x packets) will be cleared. (Backlogged packets refer to packets arrived in the previous time slots.) In fact, due to the predetermined sequence of configurations used, middle-stage port j knows beforehand which $\mathrm{VOQ}_2(j,k)$ will be cleared at which time slot.

Middle-stage port j generates the occupancy vector upon receiving the destination report from input i. The destination report contains the destinations of *all* the packets to arrive in the following slot duration. Therefore, at the time the occupancy vector is generated (in the middle of slot t), it already foresees the VOQ status at the time the last packet sent in slot t arrives at middle-stage port j (see Fig. 5). The occupancy vector is then appended to the next packet sent in the second switch fabric for transmission.

When the occupancy vector arrives at output k and is made available to input k at the beginning of slot $t+1$, the input port batch scheduler selects and sends up to x packets to middle-stage port j. It should be emphasized that the scheduling is based on what *will* happen when the selected packets arrive at middle-stage port j (i.e., the information in the occupancy vector received). Notably, the first packet from input k will arrive at middle-stage port j right after the last packet from input i. The bandwidth of switch fabric is fully utilized.

Without loss of generality, we assume a LQF batch scheduler at input port k. Specifically, input k identifies the set of VOQ_2's at middle-port j that has room for new packets; denote this set by $\boldsymbol{A}$. Find the longest queue $\mathrm{VOQ}_1(k,h)$ such that $\mathrm{VOQ}_2(j,h)$ belongs to $\boldsymbol{A}$. Then, the HOL packet at $\mathrm{VOQ}_1(k,h)$ is scheduled for sending. Update $\boldsymbol{A}$ and the size of $\mathrm{VOQ}_1(k,h)$. Then, the above process is repeated until x packets are scheduled (or no more packets are available).

Based on the staggered symmetry and in-order packet delivery properties of the joint sequence in Fig. 2(a), the delay a packet experiences at a middle-stage port is again bounded by $[1, N]$ slots. The in-order packet delivery is also ensured as every packet belonging to the same flow will always experience the same delay at any middle-stage port. In Fig. 5, when the last bit of the xth packet arrives at the middle-stage port, the first-stage switch fabric can start to reconfigure. When the last bit of the xth packet departs the second switch fabric, the second switch fabric can start to reconfigure. In other words, the reconfiguration of second fabric can start before the last bit of the xth packet arrives at the output port. For optical switch fabrics with nonnegligible amount of reconfiguration overheads [37], such a pipelined packet transmission and reconfiguration can be very efficient.

V. Performance Evaluations

In this section, the performance of our proposed feedback-based scheduling algorithms is compared with some representative algorithms by simulations. In the following, we only present simulation results for switch with size $N = 32$, although similar conclusions apply to other sizes. Both single- and multicabinet implementations of a load-balanced switch are considered. As the duration of a time slot may be different when different scheduling algorithms are used (see Figs. 3 and 5), the delay performance is measured by the number of time units, where each time unit is equivalent to the transmission time of a packet at line rate. In our simulations, three traffic models are used:

- *Uniform*. At each unit of time for each input, a packet arrives with probability p and destines to each output with equal probability.
- *Uniform bursty*. Bursty arrivals are modeled by the ON/OFF traffic model. In the ON state, a packet arrives in every time unit. In the OFF state, no packet arrivals are generated. Packets of the same burst have the same output, and the output for each burst is uniformly distributed. Given the average input load of p and average burst size s, the state transition probabilities from OFF to ON is $p/[s(1-p)]$, and from ON to OFF is $1/s$. Without loss of generality, we set burst size $s = 30$ packets.
- *Hotspot*. Packets arriving at each input port in each time unit with probability p. Packet destinations are generated as follows. For input port i, packet goes to output $i + N/2$ with probability $1/2$, and goes to any other output with probability $1/[2(N-1)]$.

A. Single-Cabinet Implementation

In the single-cabinet implementation of a load-balanced switch, the propagation delay between linecards and switch fabrics is negligible. In this case, we focus on studying the performance of the three proposed port-based scheduling algorithms in Section III-E: round-robin (RR), longest queue first (LQF), and earliest departure first (EDF). For comparison, we also implement:

- LQF with byte-focal switch architecture (LQF_Byte-Focal) [10], which outperforms FOFF and, in general, is the best performing algorithm based on resequencing buffer;
- CR algorithm [16], which is the best performing frame-based scheduling algorithm;
- iSLIP algorithm [6], which serves as a benchmark for single-stage input-queued switches. Specifically, we implement iSLIP with a single iteration (iSLIP-1), as multi-iterations involve heavy communication overhead;
- Output-queued switch, which serves as the lower bound.

Fig. 6 shows the delay-throughput performance under uniform traffic. We can see that three input port schedulers RR, LQF, and EDF yield comparable and less-than-20-unit delay performance for input load up to $p = 0.9$. When $p > 0.94$, LQF gives the best performance (as it always serves the most needed flow first), followed by EDF and RR. The average packet delay at middle-stage ports can be easily derived: $(1+N)/2 = 16.5$ time units. If we deduct this portion from the overall delay, we can see that the (input port) delay of our scheduling algorithms matches the output-queued switch performance very well. Compared to LQF_Byte-Focal, our three schedulers give significantly smaller delay. When p is reasonably large (>0.6), our algorithms also beat iSLIP-1 and CR. When $p = 0.7$, the delay of LQF_Byte-Focal is 95 time units, iSLIP-1 44, CR 152, and ours only 20.

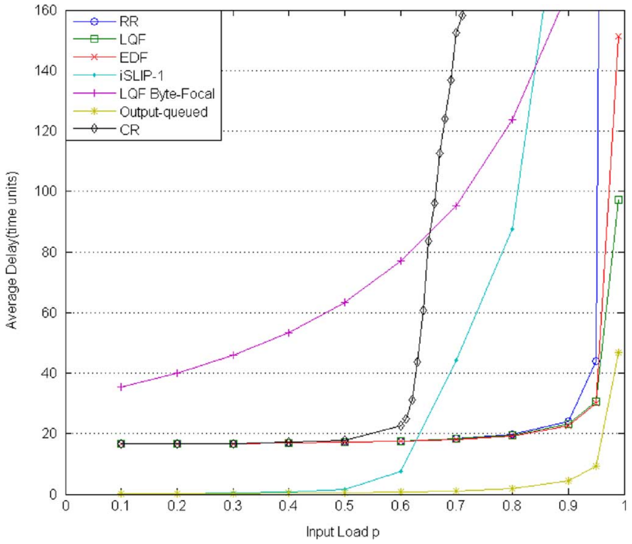

Fig. 6. Delay versus input load, under uniform traffic.

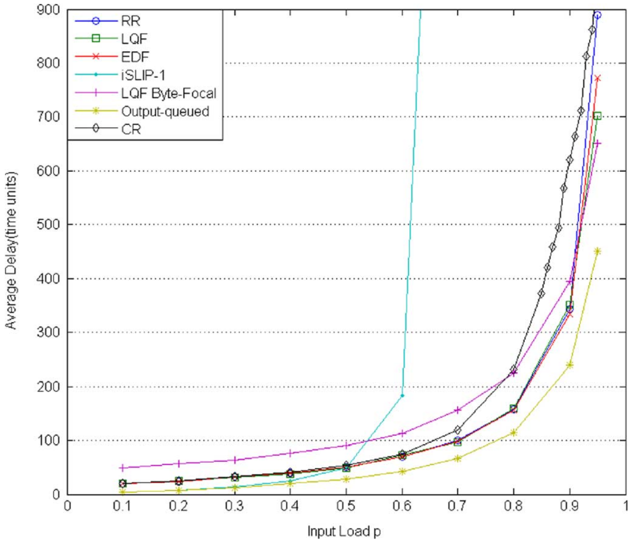

Fig. 7. Delay versus input load, under bursty traffic.

Figs. 7 and 8 show the delay-throughput performance under uniform-bursty and hotspot traffic, respectively. In Fig. 7, we can see that delay builds up quickly with input load, which is due to the bursty traffic nature. Nevertheless, our RR, LQF, and EDF still outperform LQF_Byte-Focal and CR algorithms. At $p = 0.8$, the delay of LQF_Byte-Focal is 224 time units, 232 for CR, 156 for our RR/LQF/EDF, and 114 for output-queued switch. From Fig. 8, again we can see that our three schedulers are consistently better than others. Among the three, LQF again gives the best/lowest delay performance. Nevertheless, it is interesting to point out that the performance difference among the three schedulers is much smaller than that in a single-stage switch, and this is due to the use of the first-stage switch for load balancing. For simplicity, we shall only concentrate on LQF.

B. Multicabinet Implementation

In the multicabinet implementation, we assume the propagation delay between linecards and switch fabrics is y time units,

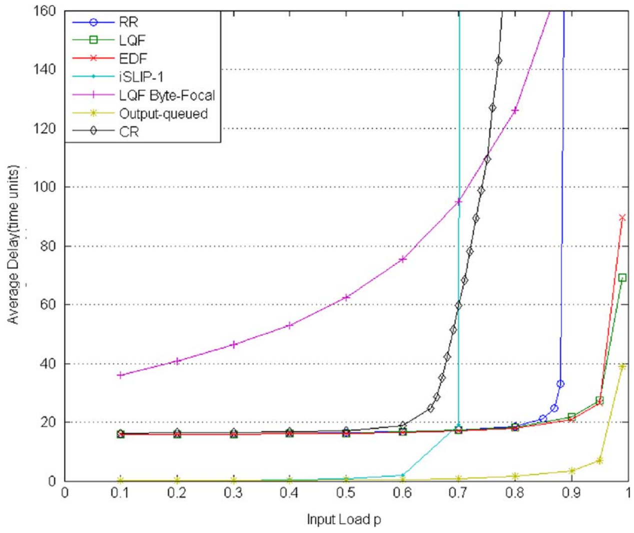

Fig. 8. Delay versus input load, under hotspot traffic.

and we vary y from 1 to 2. For simplicity, we ignore the overheads for switch reconfiguration, scheduling, etc. Three scheduling algorithms are compared:

- LQF without batch scheduling. This is a direct extension from the single-cabinet case. When propagation delay is y time units, we denote the algorithm by LQF/y. The operation of LQF/y is based on Fig. 4, where only one packet can be sent in each slot.
- LQF with batch scheduling (as in Fig. 5). When propagation delay is y, we denote the algorithm by $\mathrm{B}-\mathrm{LQF}/y$.
- SRR (Synchronous Round-Robin) algorithm [27]. When the propagation delay is y, we denote SRR as SRR/y. We regard SRR as a "generalization" of iSLIP [6] for multicabinet implementation. Therefore, SRR serves as a benchmark for single-stage input-queued switches. For more details of SRR, please refer to [27].

Note that we do not compare with LQF_Byte-Focal [10] and CR [16] because they cannot be used for multicabinet implementation.

When our B-LQF is used and the propagation delay is y time units, the number of packets that can be sent in each time slot is $2y$. From Fig. 9, we can see that due to the inefficiency caused by propagation delay, LQF/y can only obtain up to 25% and 50% throughput when $y = 2$ and 1, respectively. With $\mathrm{B}-\mathrm{LQF}/y$, close to 100% throughput can be obtained. Note that the average middle-stage port delay is still 16.5 *slots*. Since the duration of a slot is $2y$ time units, the average middle-stage port delay is 33 time units for $y = 1$ and 66 for $y = 2$.

In Fig. 10, our $\mathrm{B}-\mathrm{LQF}/y$ again yields close to 100% throughput under bursty traffic. Despite of the fact that the middle-stage packet delay increases with the slot duration, it is interesting to observe that when input load $p > 0.94$, B-LQF/2 starts to outperform B-LQF/1, though very slightly. The reason is as follows. In a time slot, each input port can send up to $2y$ packets to a middle port with B-LQF. Therefore, packets in B-LQF/2 tend to have a higher chance to enter the middle port than B-LQF/1. The earlier packets enter the middle port, the less input port delay they experience. Thus, with B-LQF/2, packets tend to experience less input port delay. Under heavy bursty loading, the input port delay dominates the overall delay

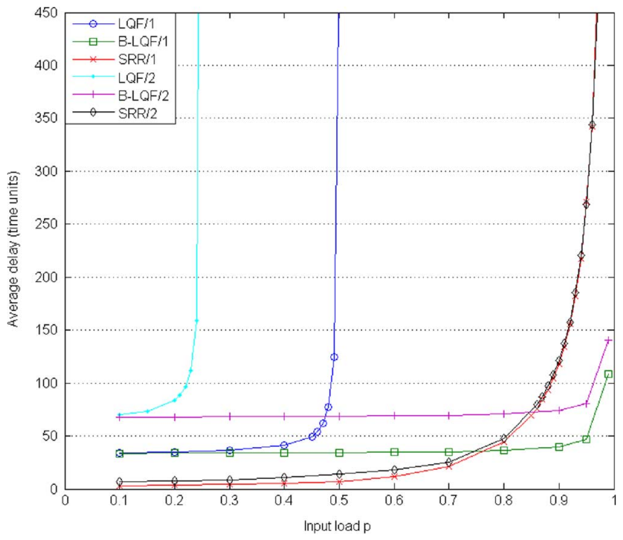

Fig. 9. Delay versus input load, under uniform traffic with propagation delay.

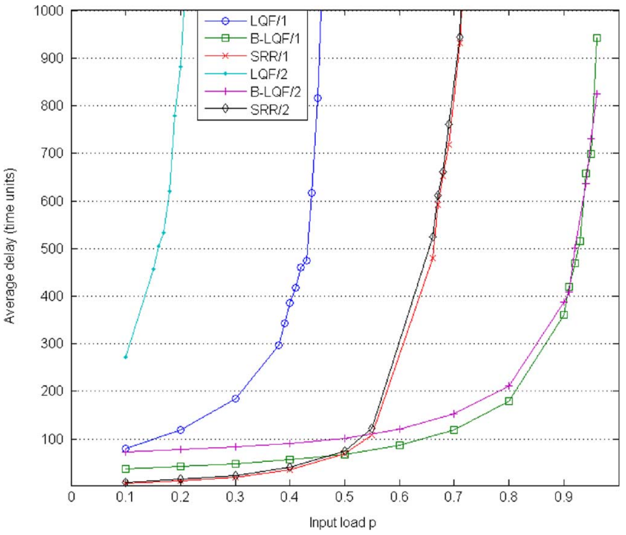

Fig. 10. Delay versus input load, under bursty traffic with propagation delay.

performance. For B-LQF/2, the drop in input port delay starts to outweigh the increase in middle port delay at $p = 0.94$. A similar performance trend is observed in Fig. 11 under the hotspot traffic.

VI. Stability Analysis

Simulation results in the previous section allow us to study the average performance under specific traffic patterns. In this section, we prove that under a speedup of two, feedback-based switch using any arbitrary work-conserving port-based scheduling algorithms (not just RR, LQF, and EDF) is stable under any admissible traffic patterns. Without loss of generality, we only focus on the case of single-cabinet implementation. For multicabinet implementation with a batch size of x packets, we can treat each batch as a single aggregate packet. Then, the multicabinet switch is equivalent to a single-cabinet switch. In other words, the propagation delay between linecards and switch fabrics does not affect/reduce the throughput performance of a multicabinet switch.

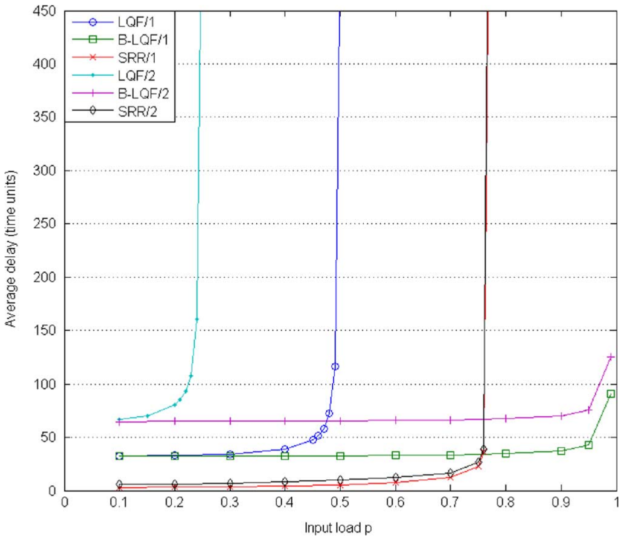

Fig. 11. Delay versus input load, under hotspot traffic with propagation delay.

A. *Constructing Fluid Model*

Like [28]–[30], we first establish a fluid model for scheduling packets. Let the number of packets in $\mathrm{VOQ}_1(i,j)$ at the beginning of time slot n be $Z_{ij}\ (n)$. Let the cumulative number of arrivals and departures for $\mathrm{VOQ}_1(i,j)$ at the beginning of slot n be $A_{ij}\ (n)$ and $D_{ij}\ (n)$, respectively. We have

$$Z_{ij}(n) = Z_{ij}(0) + A_{ij}(n) - D_{ij}(n), n \geq 0, \quad i,j = 1,\ldots,N. \tag{4}$$

Let the number of packets in $\mathrm{VOQ}_2(i,j)$ at the beginning of slot n be $B_{ij}\ (n)$. Because there is only one packet buffer for each $\mathrm{VOQ}_2(i,j)$, we have $B_{ij}\ (t) = 0$ if $\mathrm{VOQ}_2(i,j)$ is empty and $B_{ij}\ (t) = 1$ if $\mathrm{VOQ}_2(i,j)$ is occupied. The cumulative number of arrivals and departures in $\mathrm{VOQ}_2(i,j)$ at the beginning of slot n are $X_{ij}\ (n)$ and $Y_{ij}\ (n)$, respectively. The following relationship holds:

$$B_{ij}(n) = B_{ij}(0) + X_{ij}(n) - Y_{ij}(n), n \geq 0, \quad i,j = 1,\ldots,N. \tag{5}$$

We assume that the packet arrival process obeys the strong law of large numbers with probability one, i.e.,

$$\lim_{n\to\infty} \frac{A_{ij}(n)}{n} = \lambda_{ij}, \quad i,j = 1,\ldots,N$$

where λ_{ij} is the mean packet arrival rate to $\mathrm{VOQ}_1(i,j)$. The switch is, by definition, rate-stable if

$$\lim_{n\to\infty} \frac{D_{ij}(n)}{n} = \lambda_{ij}, \quad i,j = 1,\ldots,N.$$

An admissible traffic matrix is defined as the one that satisfies the following constraints:

$$\sum_i \lambda_{ij} \leq 1, \quad \sum_j \lambda_{ij} \leq 1. \tag{6}$$

If a switch is rate-stable for an admissible traffic matrix, then the switch delivers 100% throughput.

The fluid model is determined by a limiting procedure illustrated below. First, the discrete functions are extended to *right-continuous* functions. For arbitrary time $t \in [n, n+1)$

$$\begin{aligned}
&A_{ij}(t) = A_{ij}(n)\\
&Z_{ij}(t) = Z_{ij}(n)\\
&D_{ij}(t) = D_{ij}(n) + (t-n)(D_{ij}(n+1) - D_{ij}(n))\\
&X_{ij}(t) = X_{ij}(n)\\
&B_{ij}(t) = B_{ij}(n)\\
&Y_{ij}(t) = Y_{ij}(n) + (t-n)(Y_{ij}(n+1) - Y_{ij}(n)).
\end{aligned}$$

Note that all functions are random elements of $\mathbb{D}[0,\infty)$. We shall sometimes use the notation A_{ij} $(\cdot,\omega), Z_{ij}(\cdot,\omega), D_{ij}(\cdot,\omega)$ $X_{ij}(\cdot,\omega), B_{ij}(\cdot,\omega)$ and $Y_{ij}(\cdot,\omega)$ to explicitly denote the dependency on the sample path ω. For a fixed ω, at time t, we have [28]:

— A_{ij} (t,ω), the cumulative number of arrivals to $\mathrm{VOQ}_1(i,j)$;

— Z_{ij} (t,ω), the number of packets in $\mathrm{VOQ}_1(i,j)$;

— D_{ij} (t,ω), the cumulative number of departures from $\mathrm{VOQ}_1(i,j)$;

— X_{ij} (t,ω), the cumulative number of arrivals to $\mathrm{VOQ}_2(i,j)$;

— B_{ij} (t,ω), the number of packets in $\mathrm{VOQ}_2(i,j)$;

— Y_{ij} (t,ω), the cumulative number of departures from $\mathrm{VOQ}_2(i,j)$.

For each $r > 0$, we define

$$\begin{aligned}
\bar{A}^r_{ij}(t,\omega) &= r^{-1}A_{ij}(rt,\omega)\\
\bar{Z}^r_{ij}(t,\omega) &= r^{-1}Z_{ij}(rt,\omega)\\
\bar{D}^r_{ij}(t,\omega) &= r^{-1}D_{ij}(rt,\omega)\\
\bar{X}^r_{ij}(t,\omega) &= r^{-1}X_{ij}(rt,\omega)\\
\bar{B}^r_{ij}(t,\omega) &= r^{-1}B_{ij}(rt,\omega)\\
\bar{Y}^r_{ij}(t,\omega) &= r^{-1}Y_{ij}(rt,\omega).
\end{aligned}$$

It is shown in [29] and [30] that for each fixed ω satisfying (4), (5), and any sequence $\{r_n\}$ with $r_n \to \infty$ as $n \to \infty$, there exist a subsequence $\{r_{n_k}\}$ and the continuous functions $(\bar{A}_{ij}(\cdot), \bar{Z}_{ij}(\cdot)\ldots)$, where $(\bar{A}^r_{ij}(t,\omega), \bar{Z}^r_{ij}(t,\omega)\ldots)$ converges to uniformly on compacts as $k \to \infty$ for any $t \geq 0$

$$\begin{aligned}
\bar{A}^{r_{n_k}}_{ij}(t,\omega) &\to \lambda_{ij}t\\
\bar{Z}^{r_{n_k}}_{ij}(t,\omega) &\to \bar{Z}_{ij}(t)\\
\bar{D}^{r_{n_k}}_{ij}(t,\omega) &\to \bar{D}_{ij}(t)\\
\bar{X}^{r_{n_k}}_{ij}(t,\omega) &\to \bar{X}_{ij}(t)\\
\bar{B}^{r_{n_k}}_{ij}(t,\omega) &\to \bar{B}_{ij}(t)\\
\bar{Y}^{r_{n_k}}_{ij}(t,\omega) &\to \bar{Y}_{ij}(t).
\end{aligned} \tag{7}$$

Definition 1: Any function obtained through the limiting procedure in (7) is said to be a *fluid limit* of the switch. Thus, the fluid model equations using our proposed scheduling algorithms are

$$\bar{Z}_{ij}(t) = \bar{Z}_{ij}(0) + \lambda_{ij}t - \bar{D}_{ij}(t), \quad t \geq 0 \tag{8}$$

$$\bar{B}_{ij}(t) = \bar{B}_{ij}(0) + \bar{X}_{ij}t - \bar{Y}_{ij}(t), \quad t \geq 0. \tag{9}$$

Definition 2: The fluid model of a switch operating under a scheduling algorithm is said to be weakly stable if for every fluid model solution $(\bar{D}, \bar{Z})$ with $\bar{Z}(0) = 0, \bar{Z}(t) = 0$ for almost every $t \geq 0$.

From [28], the switch is rate-stable if the corresponding fluid model is weakly stable. Our goal here is to prove that for every fluid model solution $(\bar{D}, \bar{Z})$, using our scheduling algorithms, $\bar{Z}(t) = 0$ for almost every t. To prove $\bar{Z}(t) = 0$, we will use the following Fact 1 from [28]:

Fact 1: Let f be a nonnegative, absolutely continuous function defined on $\mathbb{R}^+ \cup \{0\}$, with $f\ (0) = 0$. Assume that for almost every t such that $f(t) > 0, f'(t) \leq 0$. Then, $f(t) = 0$ for almost every $t \geq 0$.

Note that $\mathbb{R}^+$ is the set of positive real numbers, and $f'(t)$ denotes the derivative of function f at time t.

B. 100% Throughput Proof

In the following, we show that our proposed scheduling algorithms give 100% throughput with a speedup of two. The result is quite strong in the sense that it holds for any arbitrary work-conserving scheduling algorithms. A work-conserving scheduler at input i can choose to serve any nonempty $\mathrm{VOQ}_1(i,k)$ for which $\mathrm{VOQ}_2(j,k)$ is empty.

Theorem 1: (Sufficiency) A work-conserving scheduling algorithm can achieve 100% throughput with a speedup of two for any admissible traffic pattern obeying the strong law of large numbers.

Proof: Let C_{ij} (t) denote the joint queue occupancy of all packets arrived at input port i, plus all packets destined for output j. We have

$$C_{ij}(t) = \sum_p \bar{Z}_{ip}(t) + \sum_m [\bar{Z}_{mj}(t) + \bar{B}_{mj}(t)]. \tag{10}$$

$\bar{Z}(t)$ and $\bar{B}(t)$ are all nonnegative, absolutely continuous functions, so C_{ij} (t) is nonnegative and absolutely continuous, too. We can see that $C_{ij}(0) = 0$, and then we have $C'_{ij}(t)$

$$C'_{ij}(t) = \sum_p \bar{Z}'_{ip}(t) + \sum_m [\bar{Z}_{mj}(t) + \bar{B}_{mj}(t)]'.$$

Combined with (8) and (9), we get

$$\begin{aligned}
C'_{ij}(t) = &\sum_p [\lambda_{ip}t - \bar{D}_{ip}(t)]'\\
&+ \sum_m [\lambda_{mj}t - \bar{D}_{mj}(t) + \bar{X}_{mj}(t) - \bar{Y}_{mj}(t)]'.
\end{aligned}$$

With a work-conserving scheduling algorithm, packets left $\mathrm{VOQ}_1(m,j)$ will enter $\mathrm{VOQ}_2(m,j)$, for $m = 1,\ldots,N$, so $\sum_m \bar{D}_{mj}(t) = \sum_m \bar{X}_{mj}(t)$, then

$$C'_{ij}(t) = \sum_p \lambda_{ip} + \sum_m \lambda_{mj} - \left[\sum_p \bar{D}_{ip}(t) + \sum_m \bar{Y}_{mj}(t)\right]'.$$

From the admissible traffic condition (6), we get

$$C'_{ij}(t) \leq 2 - \left[\sum_p \bar{D}_{ip}(t) + \sum_m \bar{Y}_{mj}(t)\right]'. \tag{11}$$

For any nonempty $\mathrm{VOQ}_1(i,j)$, i.e., $\bar{Z}_{ij}(t) > 0$, then by continuity of $\bar{Z}(t), \exists\delta$, such that $\bar{Z}_{ij}(t') > 0$ for $t' \in [t, t+\delta]$. Set

$$a = \min_{t' \in [t,t+\delta]} \bar{Z}_{ij}(t').$$

For large enough k, we have $\bar{Z}_{ij}^{r_{n_k}}(t') \geq a/2$ for $t' \in [t, t+\delta]$. Also, for large enough k we have $r_{n_k} a/2 \geq 1$. Thus, $Z_{ij}(t') \geq 1$ for $t' \in [r_{n_k} t, r_{n_k}(t+\delta)]$, which means that $\mathrm{VOQ}_1(i,j)$ holds at least one packet in the long interval $[r_{n_k} t, r_{n_k}(t+\delta)]$.

With a work-conserving scheduling algorithm, flow(i,j) packets always experience the same fixed middle-stage port delay of d slots, where d is given by (3). During the time interval $[r_{n_k} t, r_{n_k}(t+\delta)]$, when input port i is connected to any middle port g, then:

- if $\mathrm{VOQ}_2(g,j)$ is empty, a packet is transmitted from input port i to middle port g. $\sum_k D_{ik}(t)$ is increased by one;
- if $\mathrm{VOQ}_2(g,j)$ is not empty, the packet in $\mathrm{VOQ}_2(g,j)$ will be transmitted to output port j with fixed delay q, where $q = d \bmod N$. $\sum_m Y_{mj}(t)$ will be increased by one after q slots. (The packet in $\mathrm{VOQ}_2(g,j)$ will be sent when middle port g is connected to output j. If this occurs in the current time slot, $q = 0$. Otherwise, it takes another $q = d$ slots.)

If the switch is operated with a speedup of S, in a long time interval $[r_{n_k} t, r_{n_k}(t+\delta)], t' \in [t, t+\delta]$ it fulfills

$$\begin{aligned}&\sum_p \left[D_{ip}\left(r_{n_k} t'\right) - D_{ip}\left(r_{n_k} t\right)\right]\\ &\quad + \sum_m \left[Y_{mj}\left(r_{n_k} t' + q\right) - Y_{mj}\left(r_{n_k} t + q\right)\right]\\ &\quad \geq S \cdot r_{n_k}(t' - t).\end{aligned}$$

Note that $\sum_m Y_{mj}(t)$ is monotonically nondecreasing and is increased at most one in every time slot. Therefore, we have

$$\begin{aligned}\sum_m Y_{mj}\left(r_{n_k} t' + q\right) &\leq \sum_m Y_{mj}\left(r_{n_k} t'\right) + q\\ \sum_m Y_{mj}\left(r_{n_k} t + q\right) &\geq \sum_m Y_{mj}\left(r_{n_k} t\right).\end{aligned}$$

Combining them together, we have

$$\begin{aligned}&\sum_p \left[D_{ip}\left(r_{n_k} t'\right) - D_{ip}\left(r_{n_k} t\right)\right]\\ &\quad + \sum_m \left[Y_{mj}\left(r_{n_k} t'\right) - Y_{mj}\left(r_{n_k} t\right)\right]\\ &\quad + q \geq S \cdot r_{n_k}(t' - t).\end{aligned}$$

Since q is predetermined and within $[0, N-1]$, its impact is insignificant in the fluid limit [30]. Dividing the above equation with r_{n_k} and letting $k \to \infty$, fluid limits are obtained as

$$\sum_p [\bar{D}_{ip}(t') - \bar{D}_{ip}(t)] + \sum_m [\bar{Y}_{mj}(t') - \bar{Y}_{mj}(t)] \geq S \cdot (t' - t).$$

Further dividing the above equation by $(t' - t)$, and letting $t' \to t$, the derivative of the fluid limit is

$$\left[\sum_p \bar{D}_{ip}(t) + \sum_m \bar{Y}_{mj}(t)\right]' \geq S. \qquad (12)$$

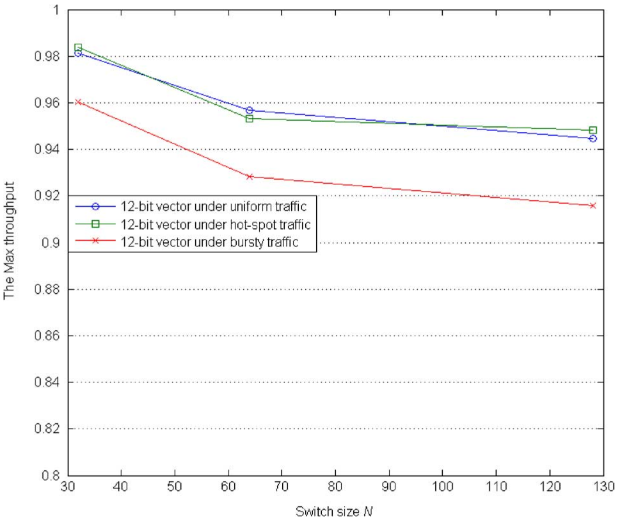

Fig. 12. Throughput versus switch size, with a fixed 12-bit feedback.

With a speedup of two (i.e., $S = 2$), combining (11) and& (12), we get

$$C'_{ij}(t) \leq 0.$$

Based on Fact 1, $C_{ij}(t) = 0$ for almost every $t \geq 0$. Due to (10) and $C_{ij}(t) = 0$, then $\bar{Z}_{ij}(t) = 0$ for almost every $t \geq 0$. Theorem 1 is proved. ■

It should be noted that existing stability proofs [8]–[17] adopt a common approach of showing that the delay performance of a specific algorithm is within a finite bound of the output-queued switch. Even under the assumption that buffer size at each middle-stage port is infinite, the derived bound w.r.t. output-queued switch is still unrealistically large (e.g., N^3 in [16]).

VII. Extensions and Refinements

A. *Cutting Down Communication Overhead*

The occupancy vector in our feedback-based two-stage switch requires N bits. To cut down the communication overhead, the size of an occupancy vector can be reduced by only reporting the status of selected middle-stage VOQs. To identify VOQs of interest, we first partition the N VOQs into u nonoverlapped sets, each identified by a set number. In each time slot, every input port piggybacks its set numbers of interest to the connected middle-stage port. This "guides" each middle-stage port to only report the status of selected VOQs.

Without loss of generality, assume the longest VOQ of each set is identified, and its length denotes the set size. Then, the top C largest sets can be identified with $C \log u$ bits, which are sent in the first switch fabric by exploiting the otherwise wasted bandwidth in the first switch fabric (see Fig. 3). For the second switch fabric, middle-stage port only needs to report the status of CN/u VOQs, thus the occupancy vector size is reduced to CN/u bits. For a switch with $N = 128, u = 32$, and $C = 3$, the occupancy vector is reduced from 128-bit to 12-bit, a saving of 90.6%. From Fig. 12, the throughput is only slightly dropped from 100% to 94.5% under uniform traffic. In Fig. 12, we have used $u = 16$ and $C = 3$ for $N = 64$, and $u = 8$ and $C = 3$ for $N = 32$.

A batch scheduler (in Section IV) can also be used to cut down the communication overhead. From Fig. 5, we can see that only a single occupancy vector of $N \log x$ bits is required for x packets sent in a time slot. The per-packet communication overhead is reduced from N bits to $(N \log x)/x$ bits.

B. Supporting Multicast Traffic

We can modify our feedback-based switch to efficiently support multicast traffic. At an input port, in addition to the N unicast $\mathrm{VOQ}_1(i,k)$'s $(k = 0, 1, \ldots N-1)$, we can add another queue $\mathrm{VOQ}_1(i,N)$ for *all* multicast packets. The input port scheduler (at input i) selects a packet for sending among its $N+1$ local queues. Priority is given to multicast traffic by examining $\mathrm{VOQ}_1(i,N)$ first. If the fanout set of its HOL packet (i.e., the set of output ports for which the multicast packet is destined) overlaps any empty VOQ_2 at the connected middle-port k, a copy of the HOL packet is sent to port k together with an N-bit duplication vector (for identifying the overlapped VOQs). Then, the fanout set of the HOL packet is updated to exclude those in the duplication vector. If the updated fanout set is empty, the HOL packet is removed from $\mathrm{VOQ}_1(i,N)$. If $\mathrm{VOQ}_1(i,N)$ is empty *or* the HOL packet of $\mathrm{VOQ}_1(i,N)$ cannot be selected (due to zero-overlap between its fanout set and any empty VOQ_2), we select a unicast packet for sending using LQF. In this case, the duplication vector is set to all 0's.

When a packet arrives at the middle-stage port, it will be cloned and stored at the (empty) $\mathrm{VOQ}_2(j,k)$'s identified by the duplication vector.

C. Amenability to Optics

In high-speed switch design, the amenability of the switch fabric to optics is essential for scalability. In [31], an optical implementation of our feedback switch leveraging WDM and arrayed waveguide grating router (AWGR) is proposed. AWGR is used because of its instinctive properties of no reconfiguration overhead and almost zero power consumption. Since full connectivity between inputs and outputs can be realized in AWGR, there are no *physical* configurations in an AWGR switch. However, the joint sequence of configurations in Fig. 2(a) can be *logically* implemented by sending packets to different outputs at different times.

It is also possible to construct an N-wavelength WDM fiber ring for connecting N linecards together. The ring network can be engineered such that the amount of time a packet should be buffered at a middle-stage port exactly matches the propagation delay that this packet experiences en route. This can ensure all-optical packet transmission from an input linecard to an output linecard.

D. Fairness Issue

For any admissible traffic patterns, as long as the switch is stable, all packets can arrive at the output ports with bounded delays. In this case, fairness in throughput is not an issue. For an inadmissible traffic pattern where some output ports are oversubscribed, the feedback switch will suffer from the ring-fairness problem, i.e., the "upstream" input ports can flood the bandwidth of an oversubscribed output port, and the "downstream" input ports will be starved. Although different input ports will have an unfair throughput share of the oversubscribed outputs, the overall switch throughput performance will not be affected if a working conserving scheduler is used.

Nevertheless, a throughput-fair scheduler is desirable, and it can be designed as follows.

- *Sending reservation request*. Each middle port j maintains two $1 \times N$ vectors, an overload vector $\{w_i\}(i = 0, 1 \ldots N-1)$, and a reservation vector $\{r_i\}(i = 0, 1 \ldots N-1)$. For any input port i, as long as the buffer for $\mathrm{VOQ}_1(i,l)$ exceeds a threshold T at time slot t, the identity of $\mathrm{VOQ}_1(i,l)$ is piggybacked (using $\log N$ bits) onto the current packet transmission to middle port j. Middle port j records $\mathrm{VOQ}_1(i,l)$ as overloaded in its overload vector $\{w_i\}$ by setting $w_i = l$.
- *Maintaining reservation status*. When middle port j connects to an output port k, middle port j examines its overload vector $\{w_i\}$. If all $w_i \neq k$ $(i = 0, 1 \ldots N-1)$, update r_k with -1, which means there is no need for reservation at $\mathrm{VOQ}_2(j,k)$. If there are any $w_i = k$ $(i = 0, 1 \ldots N-1)$, then randomly select one w_i $(i = 0, 1 \ldots N-1$ and $w_i = k)$ and set $r_k = i$. This indicates that $\mathrm{VOQ}_2(j,k)$ is reserved by input port i. Then, all w_i $(i = 0, 1 \ldots N-1$ and $w_i = k)$ are reset to -1 for next round of reservation.
- *Ensuring a reservation is honored*. Before middle port j sends its occupancy vector to output port k, j examines its reservation vector $\{r_i\}$ first. If there is any $r_i = m$, where $m > 0$ and $m \neq k$, the feedback bit in the occupancy vector for $\mathrm{VOQ}_2(j,i)$ is overwritten to 1. This is to ensure that $\mathrm{VOQ}_2(j,i)$ can only be used by input port m, which has made a reservation earlier.
- *Input port scheduling*. Any VOQ_1 sent reservation request at time slot t would be given the highest priority for scheduling at time slot $t+N$. Otherwise, the LQF scheduler is used.

Fig. 13 shows some simulation results of the above algorithm under an inadmissible server–client traffic model [32]. At each time slot for every input, a packet arrives with probability p. Linecards are partitioned into two types: a server and $N-1$ clients. The server transmits packets with equal probability to all clients. Among the $N-1$ clients, each transmits $1/3$ of its traffic toward the server and $2/3$ to the other $N-2$ clients with equal probability. When $N = 32$, the server's (linecard 0) output load q is

$$q = p \cdot (N-1)/3 = p \cdot 31/3.$$

From Fig. 13, we can see that when traffic is inadmissible (i.e., $q > 1$), using the original feedback-based switch, the delay for flow(2,0) and flow(8,0) grows quickly to infinity, whereas the delay for flow(1,0) is constant. This is because flow(1,0) is the traffic hog and it throttles the other two flows. When the proposed throughput-fair algorithm is used, the bandwidth of output port 0 is fairly shared among the three flows, indicated by their comparable delay performance.

VIII. Discussions

A. Relationship With the Work in [19]

The joint sequence of switch configurations shown in Fig. 2(c) is first adopted by the Mailbox switch [17], but its in-order packet delivery property is first discovered in [19] by modifying each middle-stage $\mathrm{VOQ}_2(j,k)$ to accommodate a

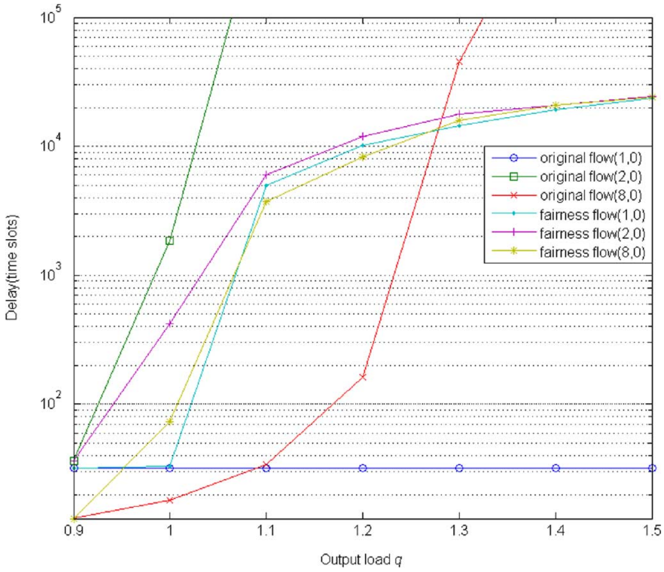

Fig. 13. Delay versus output load, under an inadmissible traffic pattern.

single packet. The ocular discovery in [19] is interesting, but cannot give much insight on why a single packet buffer per $\mathrm{VOQ}_2(j,k)$ yields a much better performance than schemes with a much larger buffer size. The proposed scheduling algorithm also makes use of the middle-stage VOQ occupancy. A dedicated feedback packet containing the middle-stage port occupancy must be sent from each middle-stage port to its connected output at the beginning of each time slot. Data packets can only be scheduled after the feedback packet arrives at an input port. Accordingly, the duration of a time slot must be large enough to account for the extra propagation delay for the feedback packet, as well as its associated packetization and synchronization overheads.

Our design differs from [19] in motivation, understanding of load-balanced switch, and the solution efficiency. In particular, we focus on properly selecting and coordinating the two sequences of switch configurations to form a joint sequence with both staggered symmetry property and in-order packet delivery property. We first reveal the essence of load-balancing in switches, i.e., neither buffer underflow nor overflow at any middle-stage VOQ. As long as there is no buffer underflow and overflow problem, the actual buffer size for each $\mathrm{VOQ}_2(j,k)$ has no impact on the switch throughput performance. However, to minimize packet delay, a small buffer size at each $\mathrm{VOQ}_2(j,k)$ is preferred because packets buffered at each middle-stage VOQ tend to experience a much longer delay than buffering them at input ports To this end, using a single-packet buffer per $\mathrm{VOQ}_2(j,k)$ is justified. To avoid middle-stage VOQ overflow and underflow, a feedback mechanism is essential. Based on the staggered symmetry property, we then devise an efficient feedback mechanism that piggybacks an occupancy vector onto a regular data packet sent. As no provision for dedicated feedback packet is required, the time slot in our design can be much shorter than that in [19].

With a single packet buffer per $\mathrm{VOQ}_2(j,k)$, we again appeal to the connection patterns in the two switch fabrics for maintaining packet order. The key is to ensure every packet belonging to the same flow to experience the same middle-stage port delay no matter the path taken by the packet. Subsequently, a family of joint switch sequences with both in-order packet delivery property and staggered symmetry property is found. Unlike [19], this gives additional flexibility of selecting an optimal joint sequence for a given traffic pattern.

B. Relationship With RRGS [35]

In [35], a pipelined RR scheduler called RRGS is proposed to achieve the maximal size matching performance as iSLIP [6] while avoiding iSLIP's iteration overhead. In RRGS, there are N pipelined schedulers. Each of them is responsible for determining a single switch configuration at the end of every N slots, or an N-slot frame. As each scheduler has unique frame starting and ending time slots, the configurations determined by the N schedulers are consecutive in time. In each frame, a scheduler visits all input ports exactly once [based on a connection pattern like (1)]. In each visit, it collects the traffic from an input port, and schedules/reserves some outputs for it on a round-robin basis. At the end of a frame, the switch is configured according to the reservations made during the frame. In [36], RRGS is extended to address the fairness issue in handling inadmissible traffic patterns and to support a variable number of schedulers.

Although the single-stage switch architecture in [35] is very different from our two-stage design, the scheduling mechanism is quite similar. We can see that the role played by a scheduler in RRGS is comparable to the role played by a middle-stage port in our design. The middle-stage port delay experienced by a packet is bounded by $[1, N]$ slots, so is the scheduling delay in RRGS (from a packet being scheduled for sending until being sent). This explains the similar delay-throughput performance yielded by RRGS in [35] and ours. However, there are also some major differences:

- RRGS requires N schedulers. Although each scheduler determines one switch configuration in every N slots, the switch configurations used in different time slots do not follow any predetermined pattern. Notably, the switch fabric in a load-balanced switch only needs to support N switch configurations, whereas for RRGS $N!$ configurations are required.
- Like iSLIP, RRGS requires rather heavy state information exchange between input ports and schedulers for generating requests and receiving feedback. It is not clear how to make this process efficient.

It is also interesting to point out that even though the concept of load-balancing is not formally recognized in RRGS, it indeed balances the incoming traffic among its N schedulers, as our feedback-based switch balances the traffic among N middle-stage ports.

IX. Conclusion

In this paper, a framework for designing feedback-based scheduling algorithms was proposed for elegantly solving the notorious packet missequencing problem of a load-balanced switch without sacrificing the switch's delay and throughput performance. Unlike existing approaches, we showed that the efforts made in load balancing and keeping packets in order can complement each other. Specifically, at each middle-stage port between the two switch fabrics of a load-balanced switch, only a single-packet buffer for each VOQ is required. In-order packet delivery is made possible by properly selecting and coordinating the two sequences of switch configurations to form a joint sequence with both staggered symmetry property and in-order packet delivery property. As compared to the existing load-balanced switch architectures and scheduling algorithms, our solutions have the modest requirement on switch hardware, but consistently yield the best delay and throughput performance under various traffic conditions.

Acknowledgment

The authors would like to thank the editor Prof. C.-S. Chang and the anonymous reviewers for their detailed and insightful comments, which helped to significantly improve the quality of the paper.

References

[1] D. Pao, N. H. Liu, A. Wu, K. L. Yeung, and K. S. Chan, "Efficient hardware architecture for fast IP address lookup," *IEE Proc. Comput. Digital Tech.*, vol. 150, no. 1, pp. 43–52, Jan. 2003.

[2] Y. Tamir and G. L. Frazier, "High-performance multi-queue buffers for VLSI communications switches," in *Proc. 15th Annu. Symp. Comput. Archit.*, Jun. 1988, pp. 343–345.

[3] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," in *Proc. IEEE INFOCOM*, San Francisco, CA, Apr. 1996, vol. 1, pp. 296–302.

[4] T. Anderson, S. Owicki, J. Saxes, and C. Thacker, "High speed switch scheduling for local area networks," *ACM Trans. Comput. Syst.*, vol. 11, pp. 319–352, 1993.

[5] N. McKeown, "Scheduling Algorithms for Input-Queued Cell Switches," Ph.D. dissertation, Univ. California, Berkeley, 1995.

[6] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Netw.*, vol. 7, no. 2, pp. 188–201, Apr. 1999.

[7] Y. Li, S. Panwar, and H. J. Chao, "On the performance of a dual round-robin switch," in *Proc. IEEE INFOCOM*, Apr. 2001, vol. 3, pp. 1688–1697.

[8] C. S. Chang, D. S. Lee, and Y. S. Jou, "Load balanced Birkhoff-von Neumann switches, part I: One-stage buffering," *Comput. Commun.*, vol. 25, pp. 611–622, 2002.

[9] C. S. Chang, D. S. Lee, and C. M. Lien, "Load balanced Birkhoff-von Neumann switches, part II: Multi-stage buffering," *Comput. Commun.*, vol. 25, pp. 623–634, 2002.

[10] Y. Shen, S. Jiang, S. S. Panwar, and H. J. Chao, "Byte-focal: A practical load-balanced switch," in *Proc. IEEE HPSR*, Hong Kong, May 2005, pp. 6–12.

[11] X. L. Wang, Y. Cai, S. Xiao, and W. B. Gong, "A three-stage load-balancing switch," in *Proc. IEEE INFOCOM*, Phoenix, AZ, Apr. 2008, pp. 1993–2001.

[12] I. Keslassy and N. McKeown, "Maintaining packet order in two-stage switches," in *Proc. IEEE INFOCOM*, New York, Jun. 2002, vol. 2, pp. 1032–1041.

[13] I. Keslassy, "The Load-Balanced Router," Ph.D. dissertation, Stanford Univ., , 2004.

[14] I. Keslassy, S. T. Chuang, K. Yu, D. Miller, M. Horowitz, O. Solgaard, and N. McKeown, "Scaling Internet routers using optics," in *Proc. ACM SIGCOMM*, Karlsruhe, Germany, Aug. 2003, pp. 189–200.

[15] J. J. Jaramillo, F. Milan, and R. Srikant, "Padded frames: A novel algorithm for stable scheduling in load-balanced switches," *IEEE/ACM Trans. Netw.*, vol. 16, no. 5, pp. 1212–1225, Oct. 2008.

[16] C. L. Yu, C. S. Chang, and D. S. Lee, "CR switch: A load-balanced switch with contention and reservation," in *Proc. IEEE INFOCOM*, Anchorage, AK, May 2007, pp. 1361–1369.

[17] C. S. Chang, D. S. Lee, and Y. J. Shih, "Mailbox switch: A scalable two-stage switch architecture for conflict resolution of ordered packets," in *Proc. IEEE INFOCOM*, Hong Kong, Mar. 2004, vol. 3, pp. 1995–2006.

[18] B. Lin and I. Keslassy, "The concurrent matching switch architecture," in *Proc. IEEE INFOCOM*, Barcelona, Spain, Apr. 2006, pp. 1–12.

[19] H. I. Lee, "A two-stage switch with load balancing scheme maintaining packet sequence," *IEEE Commun. Lett.*, vol. 10, no. 4, pp. 290–292, Apr. 2006.

[20] J. R. Nechvatal, "Asymptotic enumeration of generalised latin rectangles," *Util. Math.*, vol. 20, pp. 273–292, 1981.

[21] B. Hu and K. L. Yeung, "On joint sequence design for feedback-based two-stage switch architecture," in *Proc. IEEE HPSR*, Shanghai, China, May 2008, pp. 110–115.

[22] P. Gupta and N. McKeown, "Design and implementation of a fast crossbar scheduler," *IEEE Micro*, vol. 19, no. 1, pp. 20–28, Jan.–Feb. 1999.

[23] Y. S. Lin and C. B. Shung, "Quasi-pushout cell discarding," *IEEE Commun. Lett.*, vol. 1, no. 5, pp. 146–148, Sep. 1997.

[24] K. L. Yeung, "Efficient time slot assignment algorithms for TDM hierarchical and non-hierarchical switching systems," *IEEE Trans. Commun.*, vol. 49, no. 2, pp. 351–359, Feb. 2001.

[25] B. Wu, K. L. Yeung, M. Hamdi, and X. Li, "Minimizing internal speedup for performance guaranteed switches with optical fabrics," *IEEE/ACM Trans. Netw.*, vol. 17, no. 2, pp. 632–645, Apr. 2009.

[26] C. Minkenberg, R. Luijte, F. Abel, W. Denzel, and M. Gusat, "Current issues in packet switch design," in *Proc. ACM SIGCOMM*, Jan. 2003, pp. 119–124.

[27] A. Scicchitano, A. Bianco, P. Giaccone, E. Leonardi, and E. Schiattarella, "Distributed scheduling in input queued switches," in *Proc. IEEE ICC*, Glasgow, Scotland, Jun. 2007, pp. 6330–6335.

[28] J. G. Dai and B. Prabhakar, "The throughput of data switches with and without speedup," in *Proc. IEEE INFOCOM*, Tel Aviv, Israel, Mar. 2000, vol. 2, pp. 556–564.

[29] M. Berger, "Delivering 100% throughput in a buffered crossbar with round robin scheduling," in *Proc. IEEE HPSR*, Poznan, Poland, Jun. 2006, pp. 403–407.

[30] Y. Shen, S. S. Panwar, and H. J. Chao, "Providing 100% throughput in a buffered crossbar switch," in *Proc. IEEE HPSR*, New York, Jun. 2007, pp. 1–9.

[31] X. Wang and K. L. Yeung, "Load balanced two-stage switches using arrayed waveguide grating routers," in *Proc. IEEE HPSR*, New York, Jun. 2007, pp. 1–6.

[32] A. Bianco, D. Cuda, J. Finochietto, and F. Neri, "Multi-metaring protocol: Fairness in optical packet ring networks," in *Proc. IEEE ICC*, Glasgow, Scotland, Jun. 2007, pp. 2348–2352.

[33] A. E. Tan, "IEEE 1588 precision time protocol time synchronization performance," National Semiconductor, Application Note 1728, Oct. 2007.

[34] R. Palaniappan, Y. Wang, T. Clarke, and B. Goldiez, "Simulation of an ultra-wide band enhanced time difference of arrival system," in *Proc. Parallel Distrib. Comput. Syst.*, Nov. 2007, pp. 306–309.

[35] A. Smiljanic, R. Fan, and G. Ramamurthy, "RRGS-round-robin greedy scheduling for electronic/optical terabitswitches," in *Proc. IEEE GLOBECOM*, Rio de Janeiro, Brazil, Dec. 1999, vol. 2, pp. 1244–1250.

[36] E. Oki, R. Rojas-Cessa, and H. J. Chao, "PMM: A pipelined maximal-sized matching scheduling approach for input-buffered switches," in *Proc. IEEE GLOBECOM*, San Antonio, TX, Nov. 2001, vol. 1, pp. 35–39.

[37] B. Wu, K. L. Yeung, P.-H. Ho, and X. Jiang, "Minimum delay scheduling for performance guaranteed switches with optical fabrics," *J. Lightw. Technol.*, vol. 27, no. 16, pp. 3453–3465, Aug. 2009.

Bing Hu (S'06) received the B.Eng. and M.Phil. degrees in communicational engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 2002 and 2005, respectively.

He is currently a Ph.D. candidate with the Department of Electrical and Electronic Engineering, The University of Hong Kong. His research interests include next-generation Internet, high-speed packet switch/router design, and all-optical networks.

Kwan L. Yeung (SM'99) was born in 1969. He received the B.Eng. and Ph.D. degrees in information engineering from The Chinese University of Hong Kong, Shatin, Hong Kong, in 1992 and 1995, respectively.

He joined the Department of Electrical and Electronic Engineering, The University of Hong Kong, in July 2000, where he is currently an Associate Professor and the Information Engineering Program Co-Director. His research interests include next-generation Internet, packet switch/router design, all-optical networks, and wireless data networks.