

A Novel Feedback-based Two-stage Switch Architecture and its Three-stage Extension

Kwan L. Yeung, Bing Hu and N. H. Liu
Dept. of Electrical and Electronic Engineering
The University of Hong Kong
Hong Kong, PRC
E-mail: {kyeung, binghu}@eee.hku.hk

Abstract— The overall delay-throughput performance of a load-balanced two-stage switch hinges on its performance of load-balancing and preventing packet mis-sequencing. We show that if a two-stage switch is configured according to a carefully designed joint sequence of N switch configurations, the switch is equipped with an efficient feedback path for load balancing and can provide in-order packet delivery. In particular, the feedback path is enabled by the *staggered symmetry property* of a joint sequence, and the in-order packet delivery requires a single-packet buffer at each middle-stage VOQ. Aiming at preventing both underflow and overflow at each middle-stage VOQ, an input selects a packet for sending based on the just-in-time feedback of middle-stage VOQ status. In this paper, three simple input port-based scheduling algorithms are studied, round robin, longest queue first and earliest departure first. Simulations show that with our proposed joint sequence of configurations, the two-stage switch gives an unbeatable overall delay-throughput performance under various traffic conditions. Under the assumption that heavy flows in the incoming traffic can be properly identified, we extend our two-stage switch architecture to three-stage to further cut down the packet delay while keeping the same 100% throughput performance.

I. INTRODUCTION

A. Input-queued switches

With the continuous growth of bandwidth in fiber links, the need for building high speed switches/routers is urgent in order to keep pace with the increased transmission rate. For switches employed output queuing, each output can receive up to the maximum of N packets in each time slot, where N is the number of input/output ports. The switch fabric and output ports must operate at N times of the speed on each input link. This makes output-queued switches difficult to scale. 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. This makes input-queued switches more suitable for building high speed switches.

On the other hand, input-queued switches suffer from the well-known problem of head-of-line (HOL) blocking, meaning that a packet is held up by another packet ahead of it in the same queue but destined to a different output. This limits the maximum throughput to just 58.6% under uniform traffic [1]. To eliminate the HOL blocking, Virtual Output Queuing (VOQ) is proposed [2], where each input port maintains a separate queue for each output. That means N dedicated VOQs are maintained by each input instead of a single shared one.

A scheduler is needed to maximize the throughput of a VOQ switch. The scheduling problem is equivalent to the matching problem in a bipartite graph. In [3], it is shown that for any admissible traffic patterns, 100% throughput can be achieved by maximum weight matching. However, maximum weight matching algorithm, with a high time complexity of $O(N^3 \log N)$, becomes the performance bottleneck. 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. Basically, each algorithm consists of three scheduling phases. In request phase, each input port sends a request to every output port for which it has packet. This involves up to N sessions of point-to-multipoint communication and each output port may receive up to N requests. In grant phase, if an output port receives one or more requests, it grants to one of them based on the algorithm-specific criteria. In accept phase, if an input port receives one or more grants, it accepts one of them. Each iteration of the three-phase scheduling allows the finding of some (additional) conflict-free matching between input and output ports. The major problem with this approach is that the communication overheads scale up very quickly as the link speed and switch size increase.

B. Load-balanced two-stage switches

For an $N \times N$ input-queued switch, we define flow f_{ik} as the packets arriving at input i and destined for output k . There are up to N^2 packet flows. Let the line rate at each input/output port be R . If the traffic at the switch is uniformly distributed, the loading of each flow is equal and is bounded by R/N^2 . If the switch fabric is configured to give the same switching capacity of R/N^2 to every flow, then 100% throughput is trivially guaranteed. The simplest way to ensure each flow receives the same R/N^2 switching capacity is to configure the switch fabric using a deterministic and periodic sequence of N configurations, such that each input is connected to each output exactly once in the sequence. Such a sequence can be constructed by cyclic shifting the set of input/output connections used in each time slot. Specifically, at time slot t , input i (for $i = 0, 1, 2, \dots, N-1$) is connected to output j , where j is given by

$$j = (i + t) \bmod N. \quad (1)$$

Varying t from 0 to $N-1$ gives a sequence of N required configurations. Since this sequence is pre-determined, a scheduler for finding the best configuration on a slot-by-slot

basis is not required. This eliminates both computation and communications overheads associated with the previous algorithms [4][5][6][7].

But the actual traffic at a switch is not uniformly distributed. To this end, a novel switch architecture is proposed in [8] to convert the original non-uniform traffic into uniform based on the concept of load balancing. The resulting load-balanced switch consists of two switch fabrics in tandem (Fig. 1). The first fabric is responsible for load balancing and the second fabric is for delivering packets based on their destination ports. It turns out that if the first fabric is also configured following a deterministic and periodic sequence of N configurations (which may or may not be the same as the sequence used by the second fabric), packets arrived at the input ports can be evenly spread out to the *middle-stage ports*, the outputs of the first switch fabric and the inputs of the second fabric. In the original two-stage switch design [8], as soon as a packet arrives, it is sent to the currently connected middle-stage port. So no input port buffers are required and load balancing relies purely on the periodic sequence of the N configurations. It is proved [8] that 100% throughput can already be achieved for a broad class of incoming traffic.

Although a load-balanced two-stage switch can eliminate the scheduler, is scalable, and provides close to 100% throughput, its major drawback is that packets can be mis-sequenced when they arrive at output ports. This is because packets of the same *flow* (i.e. from the same input port to the same output port) will be distributed to different middle-stage ports and will experience different amounts of middle-stage delays there. Many efforts are made to solve this problem. They can be divided into two types, using re-sequencing buffers at output ports to re-order packets, or preventing packets from becoming mis-sequencing anywhere in the switch. For more details, please refer to Section II.

C. Our contributions

The overall delay-throughput performance of a load-balanced switch hinges on its performance of load-balancing and preventing packet mis-sequencing. In Section II, we review the existing work on load-balanced switch design with a focus on how to solve the packet mis-sequencing problem. In Section III, we argue the importance of having a scalable feedback mechanism for better load balancing. In Section IV, we show that by properly constructing and coordinating the two sequences of N configurations used in the two stages of switch fabrics, the resulting joint sequence exhibits the *staggered symmetry property*. The feedback mechanism thus constructed allows the right piece of middle-stage port occupancy information to be delivered to the right input port at the right time. We show that with the proposed joint sequence of configurations and a single-packet buffer at each middle-stage VOQ, the two-stage switch is free of packet mis-sequencing problem. (Note that with a single packet buffer at each middle-stage VOQ, an N -bit occupancy/feedback vector for each middle-stage port is sufficient.)

To maximize the delay and throughput performance of the switch, three simple port-based scheduling algorithms, round

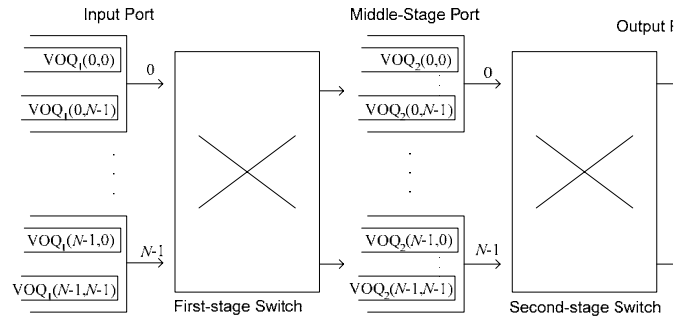


Fig. 1: A load-balanced two-stage switch.

robin. Longest queue first and earliest departure first, are designed in Section V to assist each input k in selecting a packet to send in each time slot. The basic idea is to prevent both underflow and overflow at each middle-stage VOQ. We then compare the performance of our proposed schemes with some existing algorithms in Section VI. Excellent delay and throughput performances are observed under various traffic conditions.

In Section VII, we extend the proposed two-stage switch architecture to three-stage to further reduce the packet delay. This extension is based on the assumption that we can identify heavy packet flows in the incoming traffic by, e.g. traffic measurements or from historical traffic statistics. Then we can map heavy flows to experience less middle-stage port delay using the added switch fabric at the third stage. Our proposed algorithm is resilient to errors in identifying heavy flows as the 100% throughput performance is already guaranteed by the first two stages of switch fabrics. We conclude the paper in Section VII.

II. SOLVING PACKET MIS-SEQUENCING PROBLEM

For two-stage switches, a simple approach to packet mis-sequencing is to re-order the packets at the output ports using re-sequencing buffers (not shown in Fig. 1). With the original two-stage switch architecture [8], packets can be mis-sequenced by an arbitrary amount, thus a finite re-sequencing buffer is not possible. Efforts were then made in [9][10] to bound the delay at additional costs: N writes to memory in one time slot in [9], and a very complicated re-sequencing buffer design (e.g., 3-dimensional queues) in [10].

A proactive approach is to prevent packets from becoming mis-sequencing anywhere in the switch. A major advantage is that no re-sequencing buffers are needed. Full Frames First (FFF) algorithm [11] is the first one designed this way. But due to its heavy state information exchange among switch line-cards, Full Ordered Frames First (FOFF) [12] was proposed later on to replace FFF. Notably, FOFF compromises the proactive approach by adding re-sequencing buffers to solve the mis-sequencing problem.

Another good attempt to prevent mis-sequencing is the Mailbox switch [13] in the basic form. By using a sequence of N symmetric configurations (as shown in Fig. 2(c)) in both stages of switch fabrics, a feedback path for reporting middle-

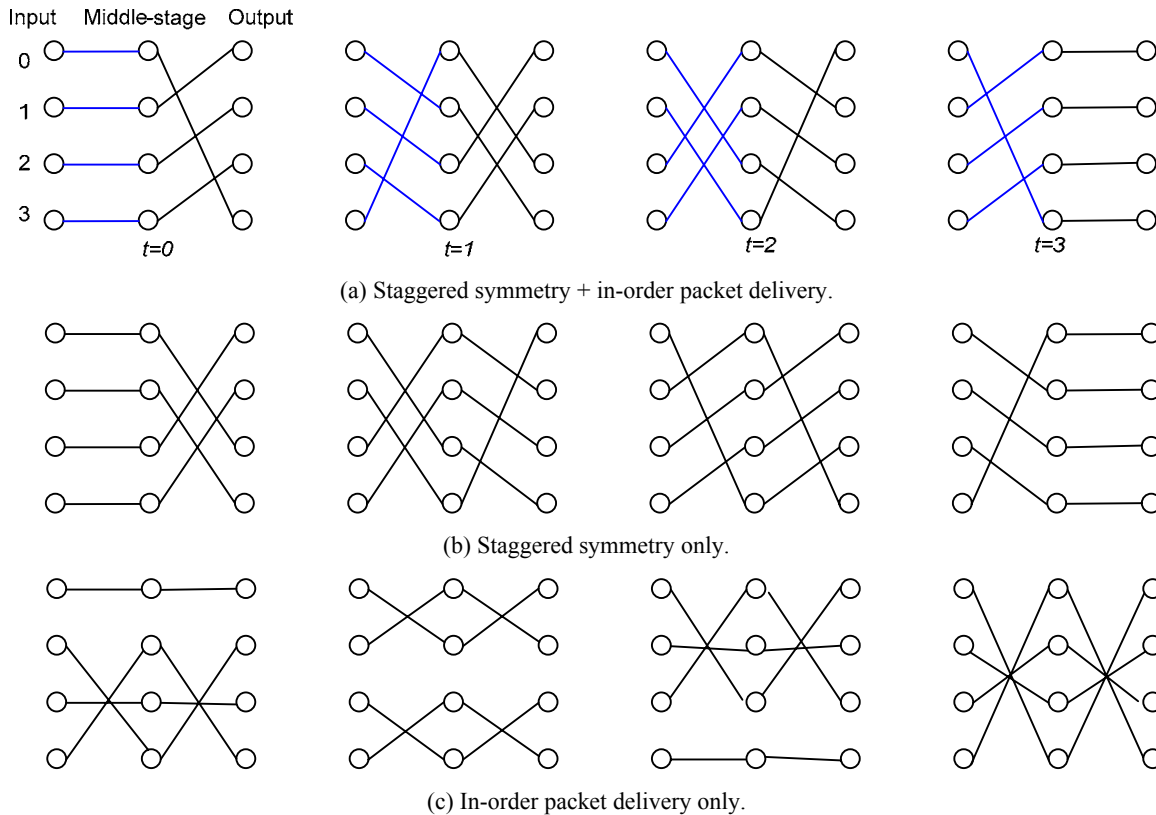


Fig. 2: Some joint sequences for a 4 x 4 load-balanced two-stage switch.

stage packet departure time is created. (More details in Section IV.D.) Based on it, the next packet in the flow will be dispatched and *inserted* in a middle-stage VOQ such that it will depart no earlier than the previous packet of the same flow. Compared with FFF algorithm, Mailbox switch trades the switch throughput ($\approx 75\%$) for simplicity. However, it was also “extended” to allow mis-sequencing for the sake of higher throughput ($\approx 95\%$). Some further extension was made in [14] for studying the amounts of buffers to be placed at inputs, outputs, and middle-stage ports.

In this paper, we show that with our proposed joint sequence of configurations and a single packet buffer at each middle-stage VOQ, the packet mis-sequencing problem is solved without sacrificing the throughput performance. We also notice that a similar result is reported in a recent paper [15]. But we differ in two aspects: a) our study is more systematic and comprehensive, and b) our feedback path design is much more efficient.

III. LOAD BALANCING & FEEDBACK

Switch performance is measured by both average packet delay and throughput. There is a general misconception that higher throughput is necessarily at the cost of poorer delay performance, and existing work mainly focuses on improving throughput at the cost of extra delay. We discuss the related issues in this section based on the two-stage switch architecture shown in Fig. 1. The switch is equipped with VOQs at both input and middle-stage ports, denoted by VOQ_1 and VOQ_2 , respectively. We use $VOQ_1(i,k)$ to represent the VOQ at input

port i with packets destined for output k . Similarly, $VOQ_2(j,k)$ is used to denote the VOQ at middle-stage port j with packets destined for output k . We define flow f_{ik} as the packets arriving at input i and destined for output k . Packets from f_{ik} are stored in $VOQ_1(i,k)$. Packets destined for output k (which may come from different input ports/flows) are placed in $VOQ_2(j,k)$ for $j = 0, 1, \dots, N-1$.

The goal of load balancing is to make the traffic seen by the second switch fabric as evenly distributed as possible. This is usually transformed into making the queue sizes of all $VOQ_2(j,k)$'s as equal as possible [10][11]. Due to the lack of feedback from $VOQ_2(j,k)$, each input port keeps track its own sending history and based on that, load balancing schemes [10][11] try to keep the difference in the cumulative number of packets that *sent* to each middle-stage port for a given flow by at most one. The problem with such a per-flow-based load balancing is that each $VOQ_2(j,k)$ is shared by *all* flows destined to output k . Since there is no coordination among different flows/inputs, the difference in $VOQ_2(j,k)$ queue sizes can be up to N packets¹. If we have a feedback mechanism that allows us to know the size of each $VOQ_2(j,k)$ before a packet is sent to it, we can do a much better job in load balancing.

We envisage that an ideal load balancing scheme should avoid both *overflow* and *underflow* at every $VOQ_2(j,k)$. Overflow is surely a bad thing as it causes packet dropping and

¹ Assume each input sends a single packet to output k via the same middle-stage port j . Then $VOQ_2(j,k)$ contains N packets while other $VOQ_2(j',k)$'s, for $j' \neq j$, have 0-occupancy.

retransmission. Underflow means that if there are packets waiting in some input ports for a particular output k , then $\text{VOQ}_2(j,k)$ should not be empty at the time that middle-stage port j (i.e. $\text{VOQ}_2(j,k)$) is connected to output k . (Note that we know in advance when a middle-stage port will be connected to a particular output because the sequence of N configurations is pre-determined.) Preventing underflow ensures the switch is work-conserving, and thus 100% throughput performance. In other words, a large buffer at $\text{VOQ}_2(j,k)$ is *not* necessary, but an efficient feedback mechanism is essential for preventing underflow and overflow at each $\text{VOQ}_2(j,k)$.

Due to the inability of accurately balancing queue sizes, per-flow-based load balancing [10][11] suffers from the underflow problem, and the problem is intensified if the incoming traffic is skewed. To ease the situation, inputs are “encouraged” to move more packets to the middle-stage ports (accordingly the buffer size of each $\text{VOQ}_2(j,k)$ ’s must be increased), in the hope to boost up the *overall* queue occupancy so as to avoid the underflow problem at a few queues. This enhances the throughput performance, as can be seen from [13][14]. Unfortunately, the delay performance is sacrificed. The poor delay performance is due to two factors. First, with the deterministic sequence of N configurations, each additional packet in an $\text{VOQ}_2(j,k)$ means this packet will experience an additional delay of N slots. We call it a delay penalty. Second, the longer the packets stay in the middle-stage ports, the more severe the mis-sequencing problem is. Consequently, a larger re-sequencing buffer/delay is required at output ports.

Indeed, if the underflow problem at each $\text{VOQ}_2(j,k)$ is solved, (extra) packets should be stored at input ports² rather than at middle-stage. This is because a) 100% throughput is already guaranteed if there is no underflow, and b) there is a delay penalty of storing packets at middle-stage ports. (Notably, similar concept is adopted in designing active queue management schemes at Internet routers, where router buffers are for absorbing bursts, rather than increasing throughput.) Then, how many buffers at each $\text{VOQ}_2(j,k)$ are required to avoid underflow? A single packet buffer is sufficient if we can dispatch a packet from an input port to an empty $\text{VOQ}_2(j,k)$ just in time. This just in time packet delivery relies on the latest status of each $\text{VOQ}_2(j,k)$, empty or not. As such, an efficient feedback mechanism must be in place for letting each input port know the latest status of $\text{VOQ}_2(j,k)$ before a packet is

² Assume we have perfect knowledge on middle-stage queue occupancy, and the sequence of configurations used in the first-stage switch fabric is obtained from (1). In time slot t , if packet A from flow f_{ik} is delivered to join $\text{VOQ}_2(j,k)$ at the *second* position in the queue, packet A has to wait for N additional slots after the HOL packet is delivered to output k , or totally $3N/2$ slots on average. Instead of dispatching packet A to $\text{VOQ}_2(j,k)$, we can keep it in $\text{VOQ}_1(i,k)$ for possible delivery to $\text{VOQ}_2(j+1,k)$ in slot $t+1$. If packet A joins $\text{VOQ}_2(j+1,k)$ at the HOL position, it can be delivered to output k in less than N slots, or $N/2$ slots on average. This is much shorter than joining $\text{VOQ}_2(j,k)$. Similarly, if joining $\text{VOQ}_2(j+1,k)$ at the HOL position fails, we can try $\text{VOQ}_2(j+2,k)$ in slot $t+2$. Repeat this process until packet A can be delivered. *Following this approach of only sending a packet to join at the HOL position of each $\text{VOQ}_2(j,k)$, each $\text{VOQ}_2(j,k)$ only needs a single packet buffer.* At the same time, there are about N chances for a packet to experience less than $3N/2$ slots in middle-stage ports.

selected for dispatching in each time slot. This feedback mechanism must be scalable, or it can easily defeat the original goal of two-stage switch.

IV. FEEDBACK-BASED SWITCH ARCHITECTURE

In this section, an efficient and scaleable feedback mechanism is enabled by a joint sequence with a staggered symmetry property. Next, we show that with a single-packet buffer at each $\text{VOQ}_2(j,k)$, the packet mis-sequencing problem is naturally solved without affecting/downgrading the switch throughput performance.

A. Constructing feedback path for load-balancing

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.

A joint sequence with staggered symmetry property is constructed in Fig. 2(a). The first stage sequence (shown in blue) is constructed from (1) by cyclic shifting the set of connections used in each slot. Each configuration in the second stage (shown in black) is obtained from the staggered symmetry property. We can see that for every pair of *staggered* configurations, e.g. the (black) configuration of the second switch fabric at $t=0$ and the (blue) configuration of the first switch fabric at $t=1$, they are mirror images of each other. Another joint sequence with the staggered symmetry property is shown in Fig. 2(b).

Since each pair of input k and output k physically reside on the same line-card in a real router implementation, the occupancy of $\text{VOQ}_2(j,k)$, for $k=0, \dots, N-1$, 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. This gives a very efficient feedback path/loop, allowing the right piece of occupancy information to be delivered to the right input port at the right time. With a single-packet buffer at each $\text{VOQ}_2(j,k)$, an N -bit vector can denote the occupancy of all N VOQs at a middle-stage port. In Fig. 2(a) and at $t=0$, middle-stage port 0 is connected to output 3. A packet is sent from $\text{VOQ}_2(0,3)$ to output 3, together with a piggybacked 4-bit vector, one for each VOQ_2 at port 0. This occupancy vector arrives at output 3 at the end of slot $t=0$, and is passed to input 3. At $t=1$, input 3 selects a packet for sending based on the vector. The associated port-based scheduling algorithm is discussed in Section V.

The detailed operation of the feedback mechanism is shown in Fig. 3. We can see that switch reconfiguration takes place in parallel with feedback relaying (from output k to input k in the same line-card) and port-based scheduling, and time for creating feedback vector at middle-stage ports overlaps with the packet receiving from input ports. On the other hand, as packet transmission in both stages of the switch occurs in parallel, a packet cannot be delivered from an input to an output in a single time slot. That is the minimum delay a packet experienced at a middle-stage port is 1 slot.

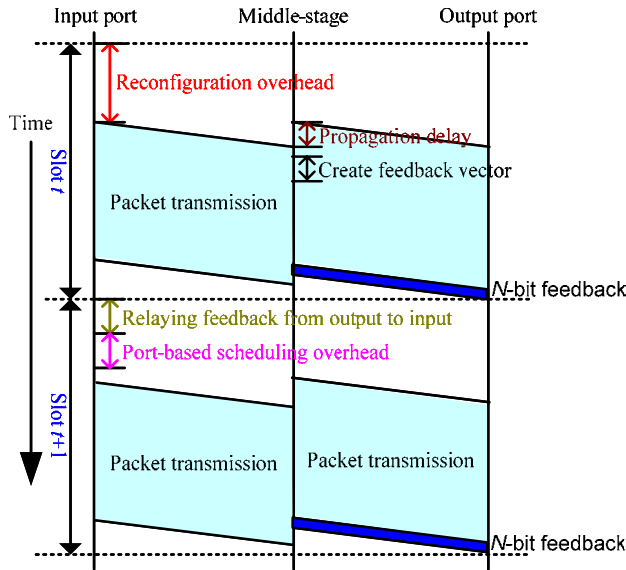


Fig. 3: Timing diagram for our proposed feedback mechanism.

B. Solving packet mis-sequencing problem

Packet order will be preserved if *every* packet of every flow experiences the same amount of delay in *any* middle-stage port. This is obviously true if middle-stage ports are bufferless, thereby rendering the same 0-slot delay for every packet passing through. Then, with single-packet-buffer-per-VOQ₂(j,k), will a load-balanced switch configured using the joint sequence with staggered symmetry property face the packet mis-sequencing problem?

Consider the two joint sequences with the staggered symmetry property in Fig. 2(a) & (b). For the joint sequence in Fig. 2(a), by observation we can see that it ensures the same middle-stage delay for every packet belongs to the same flow. Take flow $f_{0,1}$ as an example. If a packet is sent (from input 0) to middle-stage port 0 at $t=0$, it will be buffered at VOQ₂(0,1) for 2 slots until VOQ₂(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 VOQ₂(1,1) for the same 2 slots until VOQ₂(1,1) is connected to output 1 at $t=3$. On the other hand, the joint sequence in Fig. 2(b) cannot ensure in-order packet delivery. This is because for packets belong to flow $f_{0,1}$, three different middle-stage delays will be experienced, 1-slot via middle ports 0 and 2, 2-slot via port 3, and 4-slot via 1.

In this section, we focus on the joint sequence in Fig. 2(a) and prove that its in-order packet delivery property holds for any $N \times N$ switch. We know that the first stage sequence in Fig. 2(a) is constructed using (1), and the second stage sequence is constructed based on the staggered symmetry property. In fact, from Property 1 below, we can see that the general formula for determining the second stage sequence is given by (2). At time t (for $0 \leq t < N$), middle-stage port j is connected to output k , where k is given by

$$k = (j + N - 1 - t) \bmod N \quad (2)$$

If t is within $[xN, (x+1)N)$, set $t=t-xN$ before applying (2).

Property 1 (Staggered Symmetry). The joint sequence

constructed using (1) & (2) has the staggered symmetry property.

Proof: At time t , middle-stage port j is connected to output k , where k is given by (2). We need to show that at time $t+1$, if input i is connected to the same middle-stage port j , then we must have $k = i$.

At time $t+1$, we have $j=(i+t+1) \bmod N$ from (1). Substitute j into (2), we get

$$k = [(i+t+1) \bmod N + N - 1 - t] \bmod N .$$

If $(i+t+1) < N$, then

$$k = [i+t+1 + N - 1 - t] \bmod N = (i+N) \bmod N = i$$

If $(i+t+1) \geq N$, we have

$$k = [i+t+1 - N + N - 1 - t] \bmod N = i$$

Combining the above two cases, $k = i$ is true. #

Property 2 (Anchor Output). For the joint sequence constructed using (1) & (2), input i is always connected to output K , where $K = [(i+N-1) \bmod N]$, via some middle-stage port. We call K the anchor output of input i .

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

$$k = [(i+t) \bmod N + N - 1 - t] \bmod N = (i+N-1) \bmod N = K$$

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

In Fig. 2(a), input 0/1/2/3 has anchor output 3/0/1/2, via one of the middle-stage ports in each time slot.

Property 3 (Deterministic Delay in Middle-stage Ports). For the joint sequence constructed using (1) & (2), let K be the anchor output of input i . Every packet of flow f_{ik} experiences the same d slots middle-stage delay, where d is given by

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

Proof: Suppose at slot t , input i is connected to its anchor output K via middle-stage port j . If a packet (A) is successfully sent from input i to join VOQ₂(j,k), then VOQ₂(j,k) must either be empty at slot $t-1$, or the packet (B) originally in it at $t-1$ is sent to output k in slot t .

In the latter case, both packets A and B belong to the same flow f_{ik} (that's why they join the same VOQ₂(j,k)), and with the same anchor output $k=K$. It takes N slots/configurations for port j to be re-connected to output K for delivering packet A. Because of the single-packet-buffer-per-VOQ₂, packet A is the only packet in VOQ₂(j,K) and it will be delivered for sure after N slots. This gives the maximum delay of N slots.

Note that a *given* middle-stage port j is connected to each output in *descending* order of the output port numbers. This can be seen from Fig. 2(a), where middle-stage port 0 is connected to outputs 3, 2, 1, and 0 at slots $t=0, 1, 2,$ and 3 respectively. If packet A's target output k is $K-1$, middle port j will be connected to output $K-1$ after one slot, which is the minimum

delay that a packet can experience in $\text{VOQ}_2(j,k)$.

In general, if packet A's target output k is d ports away from the anchor output K (counted in descending order of port numbers), i.e. $d = K - k$ if $K > k$, and $d = K + N - k$ if $K < k$, the delay packet A experienced in $\text{VOQ}_2(j,k)$ is d slots. Obviously, the value of d is bounded by $[1, N]$. #

Assume traffic is uniformly distributed such that each input has the same input load and each packet is uniformly distributed to all outputs. The average packet delay at middle-stage ports is given by $(1+N)/2$ slots.

Property 4 (In-order Packet Delivery). If a two-stage switch is configured using the joint sequence based on (1) & (2), in-order packet delivery is guaranteed.

Proof: Assume at times t_A and t_B (where $t_B > t_A$), packets A and B of flow f_{ik} are delivered from $\text{VOQ}_1(i,k)$ and joined $\text{VOQ}_2(j_1,k)$ and $\text{VOQ}_2(j_2,k)$ respectively. Let d_A and d_B be the delays experienced in VOQ_2 by the two packets. Missequencing occurs only if packet B arrives at output k earlier than packet A, i.e. $t_A + d_A > t_B + d_B$. However, this will never happen because $t_B > t_A$ and $d_A = d_B$ from Property 3. #

Note that in order packet delivery is independent of the specific port-based scheduling algorithm used (detailed in Section V). But if more than one packet buffers are allocated to each middle-stage $\text{VOQ}_2(j,k)$, in-order packet delivery may not be guaranteed.

C. Some comments

We know that only a subset of the joint sequences with staggered symmetry property can provide in-order packet delivery. We study the necessary and sufficient conditions for a joint sequence to have both staggered symmetry and in-order packet delivery properties in another paper [18], where we show that there are $N!$ joint sequences have both in-order packet delivery and staggered symmetry properties.

Then, will the switch performance depend on the specific joint sequence adopted? Is there a way to determine the optimal sequence for a given traffic pattern? We address this issue in Section VII.

D. Related work on joint sequence design

The joint sequence in Fig. 2(c) was proposed in [13]. At time slot t , the connections between input i and middle-stage port j , and between middle-stage port j and output k , are governed by the following equations:

$$j = (t - i) \bmod N \quad \text{and} \quad k = (t - j) \bmod N.$$

The resulting joint sequence is characterized by that input k is always connected to output k via one of the middle-stage port in every time slot.

This joint sequence does not have the staggered symmetry property. But it can provide in-order packet delivery if a single-packet buffer is used at every middle-stage $\text{VOQ}_2(j,k)$. Unfortunately, [13] assumes multiple packet buffers at each $\text{VOQ}_2(j,k)$. Therefore, the in-order packet delivery property was not discovered in [13]. Under the assumption of This was discovered later in [15].

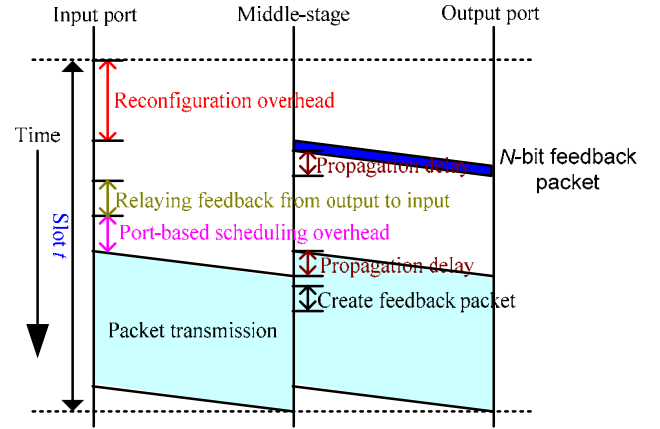


Fig. 4: Timing diagram for the feedback mechanism in [15].

Although the in-order packet delivery property of Fig. 2(c) was later revealed in [15], their adopted feedback mechanism relies on dedicated feedback packets ahead of each data packet transmission in every time slot, as shown in Fig. 4. At the beginning of each time slot, the switch is reconfigured and a dedicated feedback packet is sent from each middle-stage port to its connected output. Compared with our piggybacked feedback vector design in Fig. 3, our feedback mechanism is much more efficient³.

We also notice that the scheme in [15] allow⁴ simultaneous receiving and sending of the *same* packet at the same middle-stage $\text{VOQ}_2(j,k)$, a technique known as cut-through switching. With cut-through switching, a packet can be sent from an input to an output, via a 0-slot delay middle-stage port/queue, in the same time slot. This “gain” of 0-slot middle-stage delay is achieved at further reducing the efficiency of pipelined packet transmission in Fig. 4. That is the starting time of packet transmission in the second stage must be after the first bit arrival of the packet in the first stage. We believe this is not desirable. In this paper, we assume no cut-through switching although extending to it is trivial.

V. PORT-BASED SCHEDULING ALGORITHMS

When the feedback vector containing middle-stage-VOQ status arrives at each input port, a port-based scheduling algorithm is executed to select a packet for dispatching to the connected middle-stage VOQ. Let us first take a closer look on how the N -bit occupancy vector is constructed. In Fig. 3, this process occurs in parallel with packet receiving in the first stage switch. As the feedback is piggybacked at the end of the

³ Consider a multi-rack high-performance router such as CRS-1 from Cisco. Its line-cards are resided in different racks and are interconnected to/from the central switch fabric by fibers. Let the distance between line-cards and the central fabric be 100 m and the light speed in fiber be 2×10^8 m/s. From Fig. 4, the propagation delay for carrying the dedicated feedback back is $t_p = 500$ ns. As compared to our scheme in Fig. 3, the extra overhead in each time slot is at least 500 ns, or the bandwidth to carry a 2500-byte packet at 40Gbps. This overhead has not included the extra processing overheads for dedicated feedback packets, such as packet synchronization and O/E/O conversions.

⁴ This is not explicitly stated in [15] but can be easily derived from their simulation results.

data packet transmission in the second stage switch, it contains the occupancy changes due to packet arrival/departure in the current slot. At each $VOQ_2(j,k)$, we allow simultaneous sending the buffered packet (to an output port) and receiving another one (from an input port).

Before running the port-based scheduling algorithm, a minor adjustment to the received feedback vector is needed, as can be seen from the following example. Assume a packet from flow_{3,1} joins $VOQ_2(0,1)$ at slot $t = 1$ in Fig. 2(a). Accordingly, in the feedback vector to output 2, the bit corresponding to $VOQ_2(0,1)$ is set to 1. At the beginning of slot $t = 2$, input 2 knows that middle-stage port 0 is connected to output 1, the packet in $VOQ_2(0,1)$ is to be sent (for sure), and thus another packet can be sent to $VOQ_2(0,1)$ from input 2. Input 2 *overwrites* the just received feedback bit for $VOQ_2(0,1)$ to 0. In other words, there is no need to generate the feedback bit for $VOQ_2(j,k')$ if middle port j will connect to output k' in the next slot.

Suppose input i is connected to middle-stage port j at slot t , and the anchor output of i is K . Based on the N -bit occupancy vector piggybacked from middle-stage j in slot $t-1$ (after the proper adjustment above), we find candidate set S_j , the set of $VOQ_2(j,k)$ (for $k=0,1,\dots,N-1$) with 0-occupancy. Input i chooses the HOL packet at $VOQ_1(i,h)$ for sending only if $VOQ_2(j,h)$ is empty (i.e. $\in S_j$) and $VOQ_1(i,h)$ is not. This avoids overflow at $VOQ_2(j,h)$.

Since middle port j is connected to each output in descending order of the output port numbers, we know that in next slot $t+1$ port j will be connected to output $K-1$ (wrapped around by N). If $VOQ_2(j,K-1)$ is empty and $VOQ_1(i,K-1)$ is not, we face a possible underflow in $VOQ_2(j,K-1)$ at slot $t+1$. As such, a scheduling algorithm should always give priority to $VOQ_1(i,K-1)$ at slot t .

With the above considerations in mind, three simple scheduling algorithms are designed to select the HOL packet at a specific $VOQ_1(i,h)$.

- a) *Round-Robin* (RR): If $VOQ_2(j,K-1)$ is empty and $VOQ_1(i,K-1)$ is not, $VOQ_1(i,K-1)$ is selected (to avoid underflow). Otherwise, if $VOQ_1(i,h')$ is selected in the previous slot, then a non-empty $VOQ_1(i,h)$ is selected, where h has the smallest value to satisfy $h > h'$ and $VOQ_2(j,h) \in S_j$. **Comment:** RR gives fair access to each VOQ_1 , and is suitable for hardware implementation [6].
- b) *Longest Queue First* (LQF): If $VOQ_2(j,K-1)$ is empty and $VOQ_1(i,K-1)$ is not, $VOQ_1(i,K-1)$ is selected. Otherwise, among all the non-empty $VOQ_1(i,h)$'s with $VOQ_2(j,h) \in S_j$, the one with the longest queue size is selected. **Comment:** LQF is good for non-uniform traffic. An efficient implementation of (quasi) LQF is also available [16].
- c) *Earliest Departure First* (EDF): Among all the non-empty $VOQ_1(i,h)$'s with $VOQ_2(j,h) \in S_j$, the one with the earliest departure time at the middle-stage port is selected. (Obviously, if $VOQ_2(j,K-1)$ is empty and $VOQ_1(i,K-1)$ is not, $VOQ_1(i,K-1)$ is the queue with the earliest departure time. Note that the departure time is calculated from Property 3.) **Comment:** EDF aims at minimizing middle-

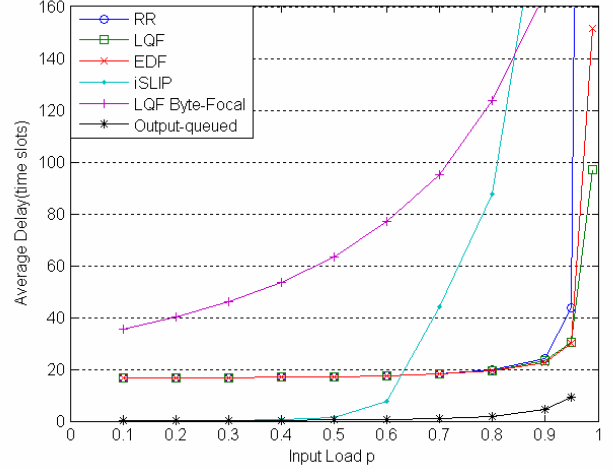


Fig. 5: Delay vs input load p , with uniform traffic

stage port delay. It may not be suitable for non-uniform traffic as input port delay is not considered.

VI. PERFORMANCE EVALUATIONS

The delay-throughput performance of our proposed feedback-based switch architecture is studied under three port-based scheduling algorithms, round robin (RR), longest queue first (LQF) and earliest departure first (EDF). For comparison, the following algorithms are also implemented: a) the LQF algorithm for Byte-Focal Switch (LQF_Byte-focal) [10], for its better than FOFF [12] performance; b) the iSLIP algorithm [6] (with a single iteration), which serves as a benchmark for single-stage input-queued switches; and c) output-queued switch, which serves as a lower bound. In this section, simulation results for switch size $N=32$ are presented. But the conclusions/observations apply for other switch sizes as well.

We do not directly compare with the scheme in [15]. This is because if we ignore the performance differences due to feedback mechanisms (by e.g., assuming the same slot duration for both schemes), then the scheme in [12] (with EDF scheduling) will give an almost identical performance with our scheme (with EDF). Besides, we have already compared the two feedback mechanisms in Section IV.D.

A. Uniform Traffic

At each time slot for each input, a packet arrives with probability p and destined to each output with equal probability. p is called the input load.

Fig. 5 shows the delay-throughput performance of all six algorithms. Among them, we can see that with our proposed feedback architecture, the three port-based scheduling algorithms RR, LQF and EDF give comparable and less-than-20-slot delay performance for input load up to $p = 0.9$. When $p > 0.94$, EDF and LQF outperforms RR. In fact, the average packet delay at middle-stage ports is 16.5 slots, as derived from $(1+N)/2$. If we deduct this portion from the overall delay, we can see that the (input port) delay of our proposed two-stage switch architecture matches very well with the ideal *output-*

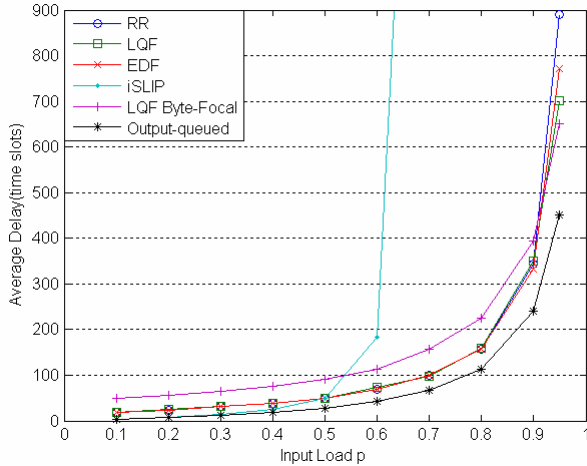


Fig. 6: Delay vs input load p , with bursty traffic

queued switch performance.

Compared with LQF_Byte-Focal, our three scheduling algorithms give significantly smaller delay. For iSLIP, our switch produces a smaller delay when p is large (>0.6). At $p=0.8$, LQF_Byte-focal requires 124 slots, iSLIP requires 88 slots, and ours only 20 slots.

B. Uniform Bursty Traffic

Bursty arrivals are modeled by the ON/OFF traffic model. In the ON state, a packet arrival is generated in every time slot. 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 $1/s$. Burst size $s = 30$ packets.

From Fig. 6, we can see delay builds up quickly with input load, which is due to the bursty traffic nature. Nevertheless, our RR, LQF and EDF algorithms generally perform better than LQF_Byte-focal and iSLIP. At $p=0.8$, LQF_Byte-focal requires 224 slots, whereas 156 for RR/LQF/EDF, and 114 for output-queued.

C. Hot-spot Traffic

Packets arriving at each input port in each time slot follow the same independent Bernoulli process with probability p . Packet destinations are generated as follows. For input port i , packet goes to output i with probability $1/2$, and goes to other outputs with same probability $1/[2(N-1)]$.

From Fig. 7, again we can see that with our three scheduling algorithms RR, LQF and EDF give comparable and consistently good performance. When traffic load $p > 0.98$, LQF outperforms EDF and RR because the most needed input VOQ queue is served first.

Combining all three sets of simulations above, we can conclude that our proposed two-stage feedback-based switch architecture is indeed very efficient. Another interesting observation is that our feedback mechanism is relatively insensitive to the port-based scheduling algorithms adopted

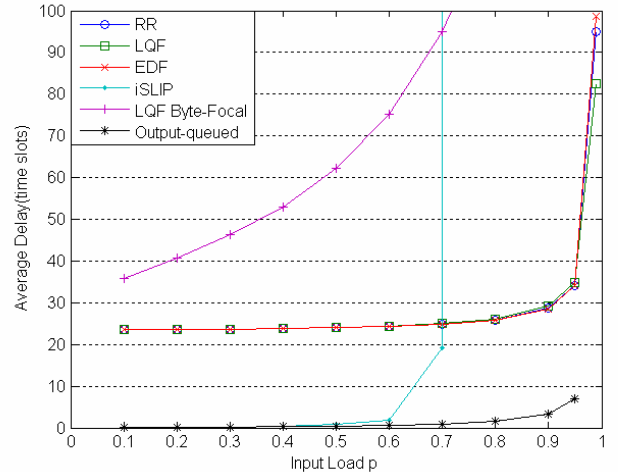


Fig. 7: Delay vs input load p , with hot-spot traffic

(unless the traffic load is very high, $p > 0.9$). Therefore, we may prefer a simple scheduler, such as round robin (RR), for its ease implementation in hardware.

VII. THREE STAGE EXTENSION

The two-stage switch is configured using the joint sequence in (1) & (2). From our discussion in Section IV.C, we know that there are $N!$ joint sequences with both staggered symmetry and in-order packet delivery properties. From Property 3 in Section IV.B, we can see that different flows experience different amounts of middle-stage delays. If we can identify heavy packet flows (e.g. by traffic measurements or from historical traffic statistics), we can cut down the average packet delay by assigning heavy flows to experience less middle-stage delay. Note that in steady state, the amount of delay a packet experiences at an input port (input delay) is independent of the middle-stage port delay.

There are two possible approaches to minimize average packet delay. First, we can examine all $N!$ joint sequences to find the one giving the lowest average packet delay. Due to the huge solution space, this brutal force approach is not practical. Besides, changing the joint sequence used by a switch in real-time may cause packet mis-sequencing during the transition period. In this section, we describe the second approach of extending the two-stage switch to three-stage as shown in Fig. 8. Our idea is to *map* those heavy flows to experience less middle-stage port delay using the additional switch fabric at the third stage. Our proposed algorithm is resilient to errors in identifying the heavy flows as the 100% throughput performance is ensured by the first two switch stages in a three-stage architecture.

In Fig. 8, the configuration of the third stage switch is designed to map heavy flows to experience less middle-stage delay. It is therefore not changed on slot-basis. It is updated only if there is a significant enough change in traffic pattern. Besides, no buffer/delay is required/incurred at the ports between the second and the third stage, or virtual output ports. Packets are directly delivered to the third stage fabric for forwarding to the (real) output ports.

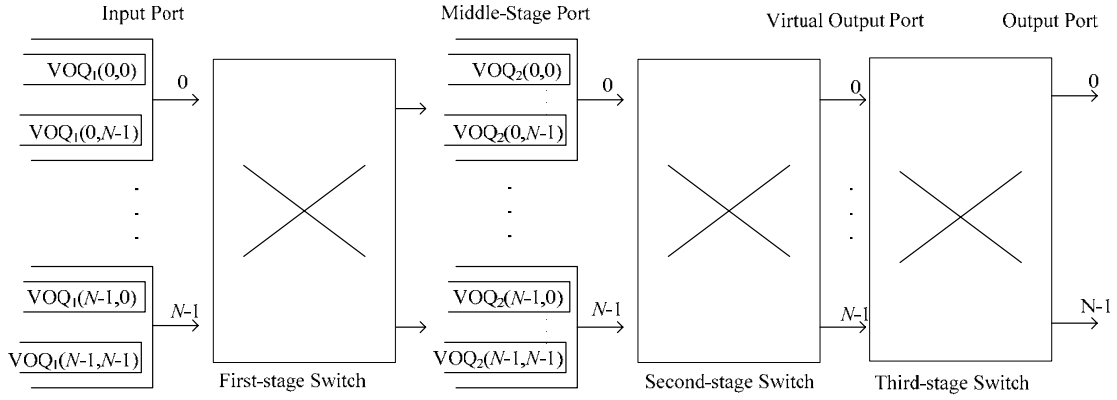


Fig. 8: A three-stage switch architecture.

Fig. 9 shows an example. Assume there is only a single flow ($\text{flow}_{0,3}$) at a 4×4 switch with the joint sequence in Fig. 2(a). The average delay of packets at middle-stage port is the delay experienced by packets of $\text{flow}_{0,3}$, or 4 slots. With the three-stage switch shown in Fig. 9(b), packets of $\text{flow}_{0,3}$ are delivered to virtual output 2 (instead of 3 in Fig. 9(a)), with a middle-stage delay of 1 slot. Then the configuration in the third stage immediately *map* packets arrived at virtual output 2 to final output 3 (with 0-delay/buffer at virtual output). The overall in-switch delay experienced by packets of $\text{flow}_{0,3}$ is reduced from 4 slots to 1.

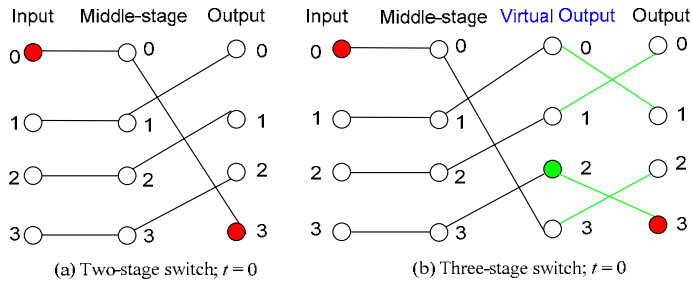


Fig. 9: A three-stage switch connection pattern.

$$\begin{bmatrix} 0.3 & 0.2 & 0.2 & 0.1 \\ 0.1 & 0.2 & 0.1 & 0.4 \\ 0.2 & 0.5 & 0.1 & 0.1 \\ 0.2 & 0.1 & 0.4 & 0.3 \end{bmatrix} \quad \begin{bmatrix} 1.9 & 1.9 & 1.9 & 2.3 \\ 2.1 & 3.1 & 2.5 & 2.3 \\ 1.9 & 1.5 & 2.3 & 2.3 \\ 2.7 & 2.1 & 2.4 & 1.9 \end{bmatrix}$$

Traffic matrix $[a_{ij}]$ Delay matrix $[d_{ij}]$

Fig. 10: Traffic matrix and delay matrix.

Assume the traffic matrix $[a_{ij}]$ is known. Then a delay matrix $[d_{ij}]$ can be constructed, where

- d_{ij} denotes that virtual output port $j-1$ is connected with real output port $i-1$, and
- the value of d_{ij} is the total middle-stage delay experienced by packets destined to output port $i-1$

For example:

$$d_{34} = 4a_{13} + a_{23} + 2a_{33} + 3a_{43} = 0.8+0.1+0.2+1.2=2.3 \text{ slots.}$$

From the resulting delay matrix $[d_{ij}]$ shown in Fig. 10, the average delay at middle-stage ports (without using the third stage) is:

$$d_{11} + d_{22} + d_{33} + d_{44} = 1.9+3.1+2.3+1.9=9.2 \text{ slots.}$$

Note that d_{ii} means virtual output i is mapped to real output i – the three-stage architecture degenerates into the original two-stage switch.

Definition: A set of entries of a matrix are *independent* if none of them occupies the same row or column.

A legitimate mapping in the third stage switch fabric must correspond to an independent set. Then minimizing the whole system delay problem becomes finding an independent set from the delay matrix such that the sum of all entries in the set is minimized. Optimal algorithms with polynomial running time exists [19,20]. For completeness, we summarize such an algorithm in Appendix-A. For the example in Fig. 10, we can find the minimum independent set to be $\{d_{13}, d_{21}, d_{32}, d_{44}\}$. The corresponding third stage connection pattern is shown in Fig. 11. The average middle-stage delay is

$$d_{13} + d_{21} + d_{32} + d_{44} = 1.9+2.1+1.5+1.9 = 7.4 \text{ slots.}$$

This gives a 19.6% improvement as compared with the two-stage switch counterpart.

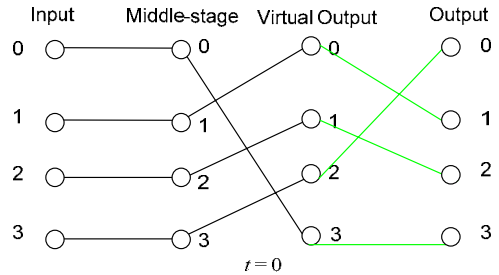


Fig. 11: Third-stage configuration based on Fig. 10.

VIII. CONCLUSIONS

In this paper, we showed that if a two-stage switch is configured according to a carefully designed joint sequence of N switch configurations, the switch is equipped with an efficient feedback path for load balancing and can provide in-order packet delivery. In particular, the feedback path is enabled by the *staggered symmetry property* of a joint sequence, and the in-order packet delivery requires a single-packet buffer at each middle-stage VOQ. Aiming at preventing both underflow and overflow at each middle-stage VOQ, an

input selects a packet for sending based on the just-in-time feedback of middle-stage VOQ status. Three simple input port-based scheduling algorithms were studied and compared in the paper. Simulations showed that with our proposed joint sequence of configurations, the two-stage switch gives an unbeatable overall delay-throughput performance under various traffic conditions. If heavy flows in the incoming traffic can be properly identified, we further showed that by extending the two-stage switch architecture to three-stage, the packet delay can be reduced while keeping the same 100% throughput performance.

With the proposed two-stage switch architecture, we can see that the packet delay at the middle-stage ports grows linearly with the switch size and is independent of traffic load. This delay performance may not be acceptable when the switch is large. As such, alternative switch architecture should be sought.

Another interesting area is to design an all-optical two-stage switch. With our proposed two-stage switch architecture, the buffer required at each middle-stage port is reduced to N packets only. This renders efficient optical buffer implementation based on fiber delay lines possible.

IX. APEENDIX-A

The following algorithm is summarized from [19,20]. For a given matrix, it finds the independent set with the minimum weight.

- a) Each row of the matrix subtracts the smallest element in this row, each column subtracts the smallest element in this column.
- b) Find a zero element, Z . If there is no starred zero in its row nor its column, mark Z with a star. Repeat for each zero of the matrix. Go to Step 3.
- c) Cover every column containing starred 0 with a line. If all columns are covered, the starred zeros form the desired independent set; Exit. Otherwise, go to Step 4.
- d) Choose an uncovered zero and mark it with a prime; If there is no starred zero Z in this row, go to step 5. If there is a starred zero Z in this row, cover this row with a line and uncover the column of Z . Repeat until all zeros are covered. Go to Step 6.
- e) There is a sequence of alternating starred and primed zeros constructed as follows: let Z_0 denote the uncovered 0'. Let Z_1 denote the 0* in Z_0 's column (if any). Let Z_2 denote the 0' in Z_1 's row. Continue in a similar way until the sequence stops at a 0', Z_{2k} , which has no 0* in its column. Unstar each starred zero of the sequence, and star each primed zero of the sequence. Erase all primes and uncover every line. Return to Step 3.
- f) Let h denote the smallest uncovered element of the matrix; it will be positive. Add h to each covered row; then subtract h from each uncovered column. Return to Step 4 without altering **any** asterisks, primes, or covered lines.

Based on the delay matrix in Fig. 10, Fig. 12 shows the detail steps in deriving the minimum independent set with the above algorithm.

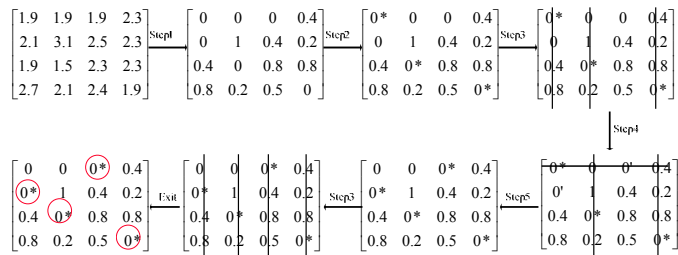


Fig. 12: An example of identifying the minimum independent set.

References

- [1] M. J. Karol, M. G. Hluchyj, and S. P. Morgan, "Input versus output queuing on a space-division packet switch," *IEEE Transactions on Communications*, Vol. 35, pp. 1347 – 1356, Dec. 1987.
- [2] Y. Tamir and G. L. Frazier, "High-performance multi-queue buffers for VLSI communications switches," *Proceeding 15th Annual Symposium Computer Architecture*, pp. 343 – 345, June 1988.
- [3] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input queued switch," *Proceeding INFOCOM*, April 1996, San Francisco, USA.
- [4] T. Anderson, S. Owicki, J. Saxes and C. Thacker, "High speed switch scheduling for local area networks", *ACM Transactions on Computer Systems*, Vol. 11, pp. 319 – 352, 1993.
- [5] N. McKeown, "Scheduling algorithms for input-queued cell switches", *PhD. Thesis*, University of California at Berkeley, 1995.
- [6] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE Transactions. On Networking*, Vol. 7, No. 2, pp. 188 – 201, April 1999.
- [7] Y. Li, S. Panwar and H. J. Chao, "On the performance of a dual round-robin switch," *Proceeding INFOCOM*, 1998.
- [8] C. S. Chang, W. J. Chen and H. Y. Hunag, "Load balanced Birkhoff-von Neumann switches, part I: one-stage buffering," *Computer Communications*, Vol. 25, pp. 611 – 622, 2002.
- [9] C. S. Chang, W. J. Chen and H. Y. Hunag, "Load balanced Birkhoff-von Neumann switches, part II: multi-stage buffering," *Computer Communications*, 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," *IEEE Workshop on High Performance Switching and Routing*, May 2005, Hong Kong.
- [11] Isaac Keslassy and Nick McKeown, "Maintaining packet order in two-stage switches," *Proceeding INFOCOM*, June 2002, New York, USA.
- [12] I. Keslassy, S.T. Chuang, K. Yu, D. Miller, M. Horowitz, O. Solgaard and N. McKeown, "Scaling the Internet Routers using Optics," *ACM SIGCOMM'03*, Karlsruhe, Germany, Aug. 2003.
- [13] C. S. Chang, D. S. Lee and Y. J. Shih, "Mailbox switch: a scalable two-stage switch architecture for conflict resolution of ordered packets," *Proceeding INFOCOM*,

March 2004, Hong Kong.

- [14] C.Y. Tu, C.S. Chang, D.S. Lee and C.T. Chiu, "Design a simple and high performance switch using a two-stage architecture," IEEE GLOBECOM 2005, St. Louis, USA, Nov. 2005.
- [15] Hyoung-Il Lee, Bhum-Cheol Lee and Seung-Woo Seo, "A load balancing scheme for two-stage switches maintaining packet sequence," IEEE ICC 2006, Istanbul, Turkey, June 2006.
- [16] Y. S. Lin and C. B. Shung, "Quasi-Pushout Cell Discarding," *IEEE Communications Letters*, Vol. 1, pp. 146-148, Sept. 1997.
- [17] K. Kar, D. Stiliadis, T. V. Lakshman and L. Tassiulas, "Scheduling algorithm for optical packet fabrics", *IEEE Journal on Selected Areas in Communications*, vol. 21, issue 7, pp. 1143-1155, Sept. 2003.
- [18] Bing Hu and Kwan L. Yeung, "On joint sequence design for load-balanced two-stage switch architecture," work in progress.
- [19] J. Munkres, "Algorithms for the assignment and transportation Problems," *J. Siam* 5, pp. 32-38, Mar. 1957.
- [20] R. Silver, "An algorithm for the assignment problem," *Comm.ACM*3, pp.605-606, Nov. 1960.